Mathematical models and methods based on metaheuristic approach for timetabling problem

Maqsood Ahmad

N° d'ordre : 2393
EDSPIC : 622

UNIVERSITE BLAISE PASCAL – CLERMONT-FERRAND II
ECOLE DOCTORALE DES SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND

THESE

présentée par

Maqsood AHMAD

en vue d'obtenir le grade de

DOCTEUR D'UNIVERSITÉ
Spécialité : INFORMATIQUE

Mathematical Models and Methods Based on Metaheuristic Approach for Timetabling Problem

---

Soutenue publiquement le 15 novembre 2013 devant le jury :

ALPAN Gulgun, rapporteur
ARTIBA Hakim, examinateur
CAUX Christophe, co-directeur de thèse
GOURGAND Michel, directeur de thèse
QUILLIOT Alain, examinateur
SOUKHAL Ameur, rapporteur

I dedicate this thesis to my beloved daughter Suha Ahmad.

AKNOWLEDGEMENT

I would like to express my gratitude to my supervisors Michel Gourgand and Christophe Cuax without their help it was impossible to complete this work. I am really thankful to them for their precious time, guidance, patience, motivation and enthusiasm. They were always available for my help whenever I found any difficulty. They were really helpful to solve my administration related problems as well.

Besides my supervisors, I would like to thank Michel Chabrolle for her care and interest in my work. Her comments and guidance played a vital role in the completion of this work.

I would also pay my gratitude to my jury members Gulgun Alpan and Ameur Soukhal for their comments and corrections which made this thesis more accurate and give us more ideas for our future work. I am grateful to my examiners for their comments and suggestions.

I would like to extend my gratitude to our Montlucon team for their support. I would never forget their cooperation. I am also thankful to my colleagues of LIMOS with whom I passed a very good time.

My profound gratitude is also for my friends, who were always with me in any time of difficulty. Their company was a source of inspiration and pleasure for me. I would always miss their company and talks.

I would like to extend my gratitude to my parents, brothers, sisters and all relatives whose prayers and good wishes were really important for me while staying abroad.

Last but not least, my profound gratitude is for my wife and daughter who helped me a lot to do this work and facilitated (especially in form of time) me during this work.

Table of Contents

List of Figures

List of Tables

SUMMARY OF THE THESIS

In this thesis we have concerned ourselves with university timetabling problems both course timetabling and examination timetabling problems. Most of the timetabling problems are computationally NP-complete problems [159, 161], which means that the amount of computation required to find solutions increases exponentially with problem size. These are idiosyncratic nature problems, for example different universities have their own set of constraints, their own definition of good timetable, feasible timetable and their own choice about the use of constraint type (as a soft or hard constraint).

Unfortunately, it is often the case that a problem solving approach which is successfully applied for one specific problem may not become suitable for others. This is a motivation, we propose a generalized problem which covers many constraints used in different universities or never used in literature. Many university timetabling problems are sub problems of this generalized problem. Our proposed algorithms can solve these sub problems easily, moreover constraints can be used according to the desire of user easily because these constraints can be used as reference to penalty attached with them as well. It means that give more penalty value to hard constraints than soft constraint. Thus more penalty value constraints are dealt as a hard constraint by algorithm. Our algorithms can also solve a problem in two phases with little modification, where in first phase hard constraints are solved. In this work we have preferred and used two phase technique to solve timetabling problems because by using this approach algorithms have broader search space in first phase to satisfy hard constraints while not considering soft constraints at all.

Two types of algorithms are used in literature to solve university timetabling problem, exact algorithms and approximation algorithms. Exact algorithms are able to find optimal solution, however in university timetabling problems exact algorithms constitute brute-force style procedures. And because these problems have the exponential growth rates of the search spaces, thus these kinds of algorithms can be applied for small size problems. On the other side, approximation algorithms may construct optimal solution or not but they can produce good practically useable solutions. Thus due to these factors we have proposed approximation algorithms to solve university timetabling problem.

We have proposed metaheuristic based techniques to solve timetabling problem, thus we have mostly discussed metaheuristic based algorithms such as evolutionary algorithms, simulated annealing, tabu search, ant colony optimization and honey bee algorithms. These algorithms have been used to solve many other combinatorial optimization problems other than timetabling problem by modifying a general purpose algorithmic framework. We also have presented a bibliography of linear integer programming techniques used to solve timetabling problem because we have formulated linear integer programming formulations for our course and examination timetabling problems.

We have proposed two stage algorithms where hard constraints are satisfied in first phase and soft constraints in second phase. The main purpose to use this two stage technique is that in first phase hard constraints satisfaction can use more relax search space because in first phase it does not consider soft constraints. In second phase it tries to satisfy soft constraints when maintaining hard constraints satisfaction which are already done in first phase.

As mentioned above, our scientific investigations are related to linear integer programming formulations for generalized university course and examination timetabling problems, transformation of course timetabling problem models to resource constrained project scheduling problem (RCPSP) and metaheuristics to solve timetabling problem. A major part of thesis consists of examining proposed algorithms. From our studies of these mathematical models and algorithms, the following scientific work is made.

• We have proposed two mathematical formulations of course timetabling problem which are the prototype of single-mode RCPSP. The main benefit of these formulations is that these give us the idea of using different durations for lessons in course timetabling and the implementation of precedence constraints through RCPSP to timetabling problem. This work equates the two different problems through mathematical formulation.

• We have proposed a new 0-1 linear integer programming formulation for university course timetabling problem. The mathematical model for the problem provides many operational rules and requirements which are needed in many universities. We have discussed how different objective functions can be deduced from this formulation by using these mathematical relations as a soft constraint.

• Different university environments have been shown as a sub problem of our generalized model. Sub problem mathematical formulations have been deduced from generic model. We have written mathematical equations of objective functions of these problems.

• A new 0-1 linear integer programming formulation for examination timetabling problem is proposed. This generic model is made by gathering many constraints from different university environments in a single problem and by adding some extra constraints which were never used in literature according to our knowledge.

•We have proposed two algorithms to solve course and examination timetabling problem. Our first algorithm is memetic algorithm. Memetic algorithm is a genetic algorithm with local search; we also have proposed a local search for our memetic algorithm. Our second algorithm is honey bee mating algorithm, use of this algorithm in educational timetabling problems is not common. We have discussed in detail differences and similarities between our algorithms and previously used algorithms.

Our thesis is structured as follows.
In Chapter 1 we have given an introduction to the university timetabling problem. We have explained what is timetabling problem and what kind of constraints are used in university timetabling problem. We have discussed similarities of graph coloring problems with educational timetabling problems. We also have given a review of different solution techniques used to solve this kind of problems, concentrating mainly on metaheuristics.

In Chapter 2, we have given a connection of single mode RCPSP with course timetabling problem. This chapter transforms course timetabling problem to RCPSP through two equivalent mathematical formulations proposed by us.
In Chapter 3, we have proposed a generic model for university course timetabling problem. We have defined different sets, sub sets and parameters to formulate a generalized mathematical model.
In Chapter 4, we have proposed a generic model for examination timetabling problem, this model resembles to the generic model for course timetabling but its structure is different from course timetabling problem.

In Chapter 5, we have discussed about different instances from literature. The formulations of these instances can be made by our generic model for university course timetabling problem. We have also discussed instances used in literature for examination timetabling problem. We have shown in detail how these instances can be obtained from our model.

In Chapter 6, we have explained in detail our proposed memetic algorithm and honey bee mating algorithm. We have explained how these algorithms work. We have explained in detail our chromosome representation, different operators used and different repair functions.

In Chapter 7, we have solved dataset instances of ITC-2007 with our proposed memetic algorithm and compared our results with the results of techniques used in literature. We have generated data for our generalized problem for both course and examination timetabling. We solve these datasets with our proposed algorithms. After getting results we have discussed about the performance of these algorithms. We have also tested the effect of different parameters and operators on the performance of algorithm.

At the end, we have given summary of the main conclusions which can be drawn from this thesis. Moreover, future research perspectives have been discussed which can give the good ideas for further possible research directions.

We want to keep this thesis at a reasonable length and we write this thesis considering that the reader has some basic knowledge of university timetabling problem and used solution techniques especially metaheuristics. Readers who do not have prerequisite knowledge are invited to study some good texts about these topics. We refer these works for basic understanding of the domain.

• One can find good articles on various different timetabling problems from the website of ASAP group (http://www.asap.cs.nott.ac.uk/?q=bibliography). This website also contains many good publications of the PATAT series (The Practice and Theory of Automated Timetabling).

• One can also find good overviews on the basic principles of metaheuristics on website http://en.wikipedia.org/wiki/Metaheuristic.

• We have given reference of some good surveys on educational timetabling problems, which can be found in the bibliography at the end of this thesis.

Chapter 1

1 INTRODUCTION

This chapter presents the general overview of timetabling problems, what actually these problems are, how much trouble they can create for scheduler and how many different ways can be used to solve them. In the next Section 1.1, we have discussed different timetabling problems used by research community and expressed how these problems have a significant effect on user's life. In Section 1.2, we have presented different types of educational timetabling problems. In Section 1.3, we have spoken about major types of constraints used in educational timetabling problems. In Section 1.4, we have a categorization of these constraints. In Section 1.5, we have brief detail of different objective functions used in literature. In Section 1.6, we have talked about different neighbourhood structures used for educational timetabling as a part of different metaheuristics. In Section 1.7, we have shown a comparison between simple timetabling problem and another combinatorial optimization problem: graph colouring. In Section 1.8, we have included a detailed review of different methods used to solve various university timetabling problems proposed by researchers in the literature. This chapter's conclusion is given in Section 1.9.

1.1 Brief introduction of timetabling problems

Wren [117] defines timetabling as follows: ''Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives''.

Timetables have become very important part of our daily life such as transport, work, sports and education. It is very hard to imagine and organize life in modern society without them. The construction of workable and attractive timetables is not an easy task in real world cases where one has limited resources like people, space and time. Using limited resources it becomes a challenging problem even for the experienced designers. These timetables have a significant effect on the people who use them. These timetables construction should be as best as one can. Often such type of timetables will be updated or completely rescheduled. School

or university timetables will be rescheduled at the beginning of the new academic year, similarly bus or train timetable will be modified to cope with new road layouts and stops. So this is a problem which people and institution will face on regular basis.

Train or railway timetabling is the planning of arrivals and departures of trains at stations and from stations. The objective is to minimize travel times, satisfy user requirements and solve conflicts between different trains. This is the simple theoretical view but in real situation there are many more problems to solve. There could be many additional constraints like when some trains are standing on tracks, some platforms are not available etc. Bus or train timetables show that when a journey will be conducted on any particular routes or paths. It does not tell about which driver or which vehicle will be used. The allocation of derivers and vehicles is the part of scheduling process. So railway scheduling means design of pattern journeys [5, 6]. Employee timetabling is very important when one wants to produce goods to meet some demand in any production system. This type of schedule will take into account the availability of the human resources. So with production schedule an employee schedule should be made simultaneously [7]. Employee timetabling will minimize labor costs while production scheduling should insure that production can be done on time.

Nurse rostering problems will arise in hospital practice. This is about personal scheduling and this generates daily schedules for nurses by assigning shifts according to their skills and predefined hard and soft constraints. It can be defined this way " Given a set of shifts and set of nurses over a certain period, assign each shift of nurse subject to a set of constraints" [8]. The main purpose of this scheduling is to make a plan for nurses who can satisfy nurses and patients. Due to many constraints of conflicting nature and requirements (like coverage demand, consecutive demands of shifts, day off requirements, work load of nurses, weekend related requirements) this is a very difficult problem to solve for hospital's personal managers and for researchers as well.

Sports have great attraction all over the world. Professional sport leagues invest a lot of money in players. Now these games have become multimillion dollar industries. Big events like the Football world cup, Olympic Games and major golf tournaments generate huge television audiences worldwide. Thus the key aspect of this type of scheduling is the assignment of venues and times to different competitions over multiple venues that can satisfy

to all those who have interest in these competitions. So constraints of the problem can vary from competition to competition. These requirements could be of different type, each team should play at each station, each team will counter against each other team at least once in tournament etc. [14].

1.2 Main categories of educational timetabling problems

Educational timetabling problems could be divided in main three categories, High school timetabling, University timetabling and Examination timetabling problems. There are some good reviews on the topic in the literature [97,118,119, 73].

1.2.1 High school timetabling

In high school timetabling each class has a particular lessons and this timetabling is majorly driven by curriculum. Number of hours of each subject per week set nationally. Each class normally has a set of students which will attend same courses all day and a specific teacher will be allocated for each course. Teachers are normally allocated before scheduling so the main purpose is to assign courses of teachers to classes in particular periods.

1.2.2 University timetabling

University timetabling can be formally defined in this way, task of assigning a number of events such as lessons, examinations, meetings and so on, to a limited set of periods (perhaps rooms) in accordance with a set of constraints.

The university course timetabling problem is an assignment of courses to periods and rooms like high school timetabling but major difference between university course timetabling and high school timetabling is that university classes can have common students while school classes have disjoint sets of students. So if two classes have common students these can not be scheduled together in the same period. School teachers most of the time teach more than one class while university professor may teach only one course. In high school timetabling rooms are not a big issue because each class has its own room but in university course timetabling problem rooms have a lot of importance. In university environment some classes need some specific rooms with specific equipment. Room size plays a vital role, one big room can be partitioned into many small rooms and some rooms are not available all times.

1.2.2.1 Characteristics of university timetabling

The university timetabling problem has special features that highly depend upon the resource arrangement, their availability and other characteristics that the courses taught in a university carry along. A course can have particular requirements, which can be related to periods, teachers, rooms, sessions, arrangement of lessons of the course etc. [12].

1.2.2.2 Structure of university courses

A course offered by any institution may consist of just lessons, or lessons and recitations, or lessons, recitations, and/or work in a laboratory. The lessons are delivered by professors, teachers or other teaching staff and it is their choice, how they will carry out them in single or multi period session. Recitations are considered as lessons delivered by the same teacher, while in other cases they may be considered time for exercises to enhance the understanding of the material covered during lessons. In this last case the class is split into many parts and one or more persons are assigned to attend them. Lab work is usually part of a given course or sometimes a course by itself and the group of students is split into several sub-groups for training. Lab work is performed in special type of rooms (special type rooms may be laboratory, computer room etc.) which may need special equipment and professors and assistants are assigned to pursue these lessons. If number of students attending a course is more than a certain limit, course can be divided into sessions, these sessions can be taught by the same teacher and by different teachers depending upon the situation, resources and requirement of the university.

1.2.2.3 Types of university courses

In many universities courses are categorized in two categories mandatory and electives [12]. Mandatory courses are compulsory courses which each student should take for his basic training. These courses are more in lower grade years, so these are designed for all students of the same year. Elective courses are less in numbers in lower grade years. So these elective and mandatory courses have common students thus they should be scheduled on different periods.

In higher grade years case is opposite where mandatory courses are less in numbers but elective course are more in numbers. It is common in higher grade years to divide their students in different divisions according to student choices and each division has their own mandatory and elective courses. Then to make schedule for them is hard because some divisions have no common interests and some divisions are sharing courses. Different

students may belong to different lab groups or recitation and this will enhance more difficulty in scheduling.

1.2.2.4 Availability of resources

For the university timetabling problem resources refer to human resources, available periods and classrooms. Availability of human resources is restricted because due to administrative duties and some other projects teachers may not be available all the time. Some faculty members may like some free days or periods during the week. Similarly periods are also not available all the day, scheduler has to spare periods for lunch time or for tutorials etc., some periods are scheduled already due to the choices of faculty members to teach course in a specific period and some periods are forbidden for some courses due to non availability of teachers. Rooms may be of many types and which course uses which type of room depends on course and room type. All rooms will not be vacant all the time because some other departments of the university may be using them for their courses and these are available on specific periods for scheduling.

1.2.2.5 Rules for timetabling in the university

These basic rules in university timetabling regarded as hard constraints.

1. There is a conflict in a timetable when two or more courses are scheduled at the same period for the same teacher or for the same classroom for the same group of students. Also, when two or more teaching persons are assigned to the same group of students to teach two different courses at the same period; and lastly when two or more classrooms are assigned to the same course and to the same group of students at the same period.
2. A timetable is complete when total number of hours of the course should be scheduled, also teaching load of the teacher should not be more than a certain predefined limit. Similarly, there are rules which are regarded as soft constraints; some of them are given as follows:

1. Preferences for teachers in specific time intervals are satisfied if possible. Each faculty member can express his/her preference for his/her course. For example one does not want to teach course on Friday and other prefer to teach his/her courses in morning session.
2. Students' schedule should be as compact as possible; however there should be empty periods for lunch. So called difficult courses should be scheduled earlier in the day then others should be scheduled. If one lesson of a course is scheduled in a day then if any other lesson of the same course is assigned to the same day, it should be compact with previous one.

3. If one course is scheduled in any room first time, it should be scheduled in the same room next time.

1.2.3 Examination timetabling

The examination timetabling is an assignment of events where now events are examination. It is same as course timetabling but has few differences with course timetabling problem. Normally each course has one examination. Major difference of this timetabling with course timetabling is that if a lesson is scheduled in a room then this room will not host any other lesson in that period. But in examination timetabling a room can host more than one examination depending upon the capacity of the room and the availability of invigilator. In course timetabling one teacher can teach only one lesson per period but in examination timetabling one examination can be supervised by more than one invigilator. Restrictions are strict in examination timetabling like we can think that a student is enforced to skip his course due to overlapping in schedule but it is not possible to skip examination. In course timetabling, scheduler makes a timetable for a week and repeats it for other weeks, means that course timetabling has fixed set of periods but in examination timetabling problem number of used periods can be relaxed.

The timetabling problem, like many others in the area of combinatorial optimization, has been approached by several well-known techniques of the operational research and the computer science fields. Several surveys on course timetabling [130, 10], which focus other aspects of the problem also have managed to record this work in a systematic way by categorizing the different variations of the problem and solution approaches.

1.3 Different types of constraints

In university timetabling, a set of lessons would be scheduled into rooms and periods subject to constraints that are usually divided in two categories, which are hard and soft constraints. Constraints and their importance however differ significantly among countries and institutions. A timetable is considered to be effective when it is useable and may be considered satisfactory by the institution when it carries certain quality characteristics that keep its users satisfied at least to a certain degree.

1.3.1 Hard constraints

Hard constraints must be strictly satisfied and have higher priority than soft constraints. A time table will be called feasible if it satisfies all hard constraints. Event clash constraints perhaps are considered common hard constraints in educational timetabling problems. These constraints normally occur when there is a resource which is only one and is required by two events at the same period which is impossible. This constraint exists in almost every educational institution, for example courses taken by some common students could not be scheduled in the same period.

1.3.2 Soft constraints

Soft constraints fulfillment is not obligatory like hard constraints but these are the constraints which scheduler wants to obey if possible, but these will decide the quality of the timetable according to the timetabling policies of university concerned and by the users who will use these timetables (like students, teachers etc.).

1.4 Categorization of constraints into five main classes

Despite the wide range of these different constraints, these constraints can be categorized into five main classes. This classification was suggested by Corne et al. [11].

1.4.1 Unary constraints

These are the constraints which involve only one event. For example event c should be scheduled on Monday or event c should be scheduled on any last period of the day. This type of constraints can occur both in course and examination timetabling. It can be both hard and soft.

1.4.2 Binary constraints

These are the constraints which involve pairs of events. For example event c should be scheduled before event d or two events c and d can be scheduled in same period. This type of constraints can occur both in course and examination timetabling. It can be both hard and soft.

1.4.3 Capacity constraints

These constraints are about capacity of rooms. If a course *c* is scheduled in a room then the number of students taking course *c* should be less than or equal to the capacity of the room. This type of constraints can occur both in course and examination timetabling. It can be both hard and soft.

1.4.4 Event spread constraints

This is a constraint which talks about spreading out or clumping together events. There are some time requirements that lessons of a class should be scheduled together or examination of a class should be scheduled after a gap of g periods. So these constraints can be in both course and examination timetabling and these are normally soft constraints.

1.4.5 Agent constraints

These constraints are about the preference of the people who will use this timetable, for example teacher *a* wants two free mornings, some classes want their language courses should be scheduled in evening session, these constraints are used normally as soft constraints.
From this whole analysis it is clear that each institution has a wide range of their own preference and requirements which can entirely be different from other institutions. It is entirely understandable that each institution has their own needs and timetabling policies, for example one institution wants that course should be clump together because institution have some more courses to schedule for part time students and other institution wants that events of a student should be spread out within the week as much as possible, so these two scenarios are entirely different than each other.

If we look timetable from an optimization perspective, it tells us that if one wants to get a workable timetable in a limited time, this will depend upon the problem instance. There are some universities which have fairly loose requirements, have a lot of resources (like many rooms, less classes, less courses, etc.) and fewer events to schedule. In this scenario it is easy to find many workable solutions from the search space. On the other hand, some universities may have more demanding requirements, a huge work load and fewer resources in comparison with requirements, so in this scenario there will be very less workable solutions in the search space of the problem instance. In practice, sometimes constraints of the problem

are so complex and are making problem impossible to solve that scheduler has to relax some of them. So to solve harder problems we need some robust and powerful methods for tackling these sorts of problems.

1.5 Objective Functions

Different types of objective functions had been used in literature. In this section we shall write some of them which were used commonly by different authors. These objective functions were used for course timetabling problem.

- Maximise cost which is the measure of desirability of teachers or maximize the satisfaction degree of teachers
- Maximize total sum of rooms and time slots preference values
- Minimize the total number of non-assigned subjects
- Minimize the weighted sum of penalties of soft constraints or minimize the violation of soft constraints
- Minimize the assignment cost, the maximum total assignment at undesired time slots simultaneously
- Timetable of each class should be compact

These kinds of objective functions had been repeated in many articles. Most of the time composite objective function with the minimization of weighted sum of penalties of soft constraints had been used, so objective function would mainly depend upon these soft constraints.

Some mostly used objective functions for examination timetabling problems are described as follows.

- To spread out examinations over the exam period for each student
- Minimize the distance between rooms of an exam that is being held in multiple rooms and to minimize splitting an exam over several rooms
- Minimize the assignment cost and penalty for exceeding the maximum work load of invigilators
- Minimize the cost function

Similar to course timetabling problems, exam timetabling problems had also been used objective functions with weighted sum of penalties of soft constraints.

Next section talks about neighbourhood structures in detail because we have used neighbourhood based algorithms for our experiments.

1.6 Neighbourhood Structures

One of the most important features of a local search algorithm is the definition of its neighbourhood. In local search procedure new solution $X \oplus mv$ is obtained by using a move *mv* to a candidate solution X. Thus a neighborhood N of X can be defined by, $N(X) = \{X \oplus mv / mv \in M(X)\}$, where M(X) is the set of all possible moves which can be applied to X.

We can find many type of neighbourhood used for timetabling problem [1].

1.6.1 Simple swap

Some authors have used simple swap, which have further two types. First one swaps any lesson from to any null slot (Figure 1.1b) and the second one swaps course c_1 to the place of c_2 and c_2 to the place of c_1 (Figure 1.1 a). In Figures 1.1 to 1.3, P represents period and R represents room.

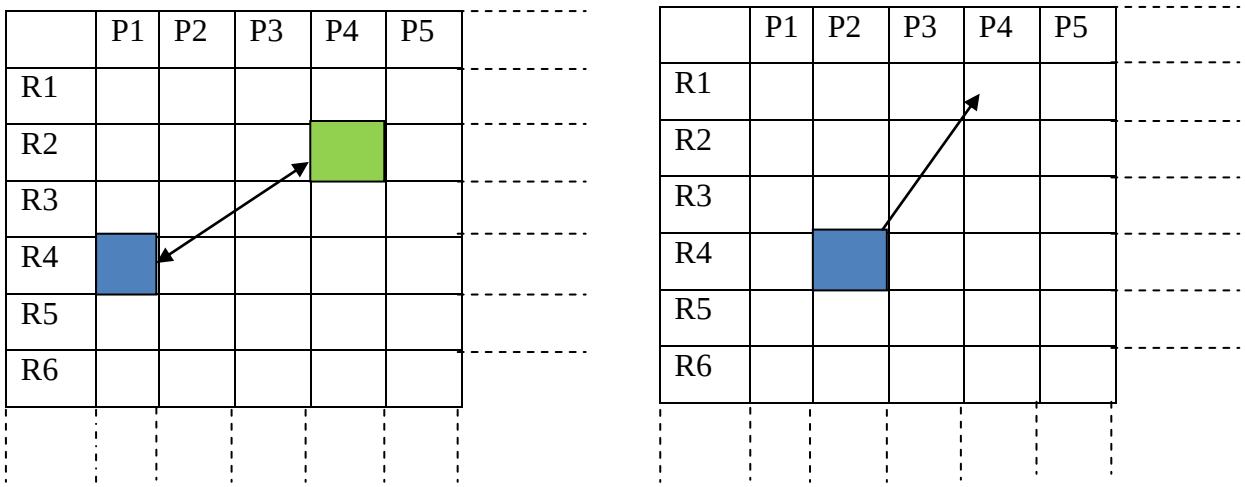


Figure 1.1: a and b: Illustration of simple swap

1.6.2 Room swap neighborhood

This is neighborhood where the move would take place in a same period, means that lessons of one period will swap with each other. If presentation is a matrix of rooms (R) and periods (P), periods are the columns of matrix and rooms are the rows of the matrix (Figure 1.2). Then the lessons will swap in the same column. Benefit of this type of move is that after getting feasible solution, it will not become infeasible.

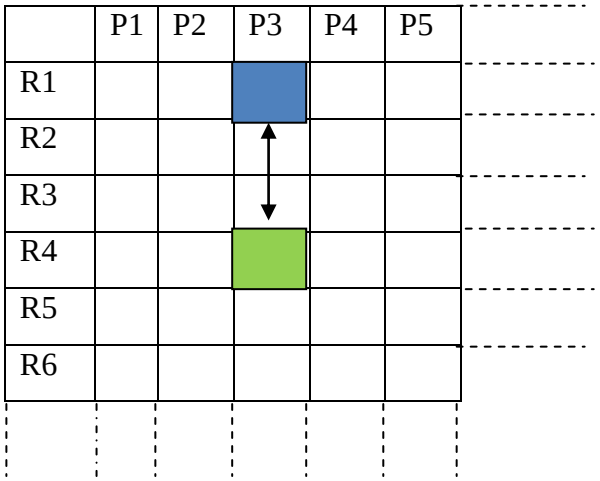


Figure 1.2: Illustration of room swap neighborhood

1.6.3 Time swap neighborhood

This explores the moves that select a lesson and swap it horizontally (Figure 1.3), this means that chosen lesson will use same room but in any other period. By using this one is becoming much more specific and it will put a restriction on search space. It can be used partially as neighborhood but overall it is not advisable. The simple swap move is the generalization of time and room move where we can swap a lesson from any slot to any other slot without focusing on any room or period.

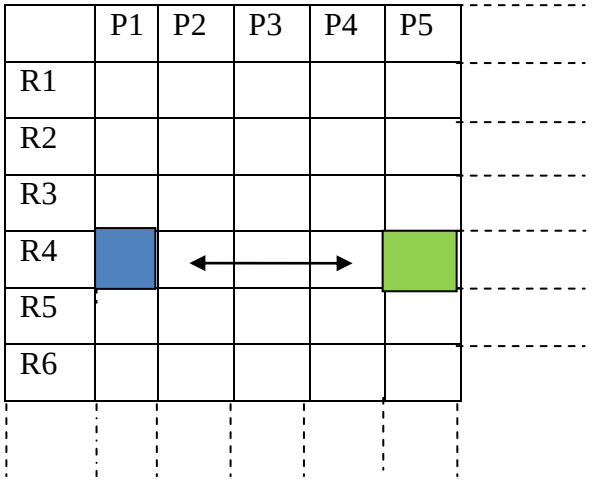


Figure 1.3: Illustration of time swap neighborhood

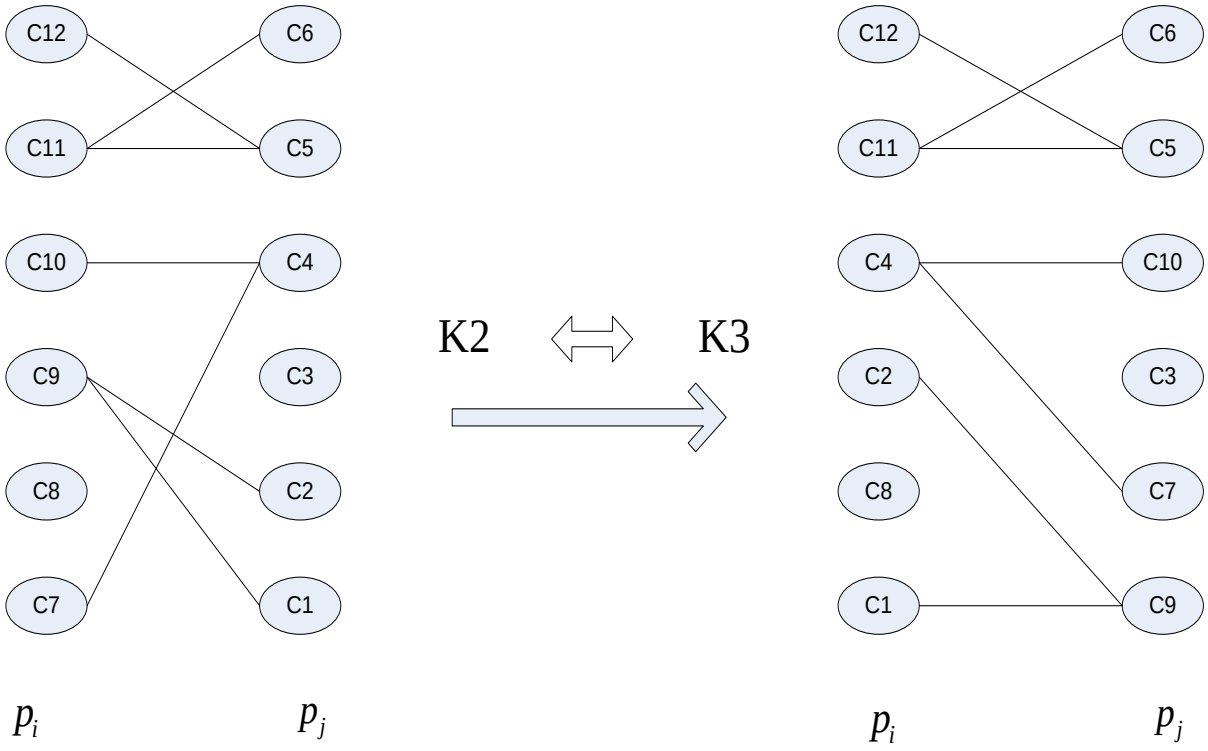
1.6.4 Kempswap neighborhood

This move was defined by Zhipeng Lü and Jin-Kao Hao [1] and is called kempswap. A kemp swap is the interchange of two kemp chains. Course can be seen as lessons and conflicts and we can write them in form of a graph. Nodes are courses and edge between two courses is the conflict between these courses. This means that these courses could not be scheduled in a same period. A kemp swap produces a feasible assignment by swapping periods assigned to courses belonging to one or two specified kemp chains.

Let K_1 and K_2 be two kemp chains in the subgraph of two periods p_i and p_j. A kempswap produces an assignment by replacing p_i with $(p_i \_ (K_1 \cup K_2)) \cup (p_j \cap (K_1 \cup K_2))$ and p_j with $(p_j \_ (K_1 \cup K_2)) \cup (p_i \cap (K_1 \cup K_2))$. According to definition at least three courses should be involved.

We can illustrate the procedure by Figure 1.4 which is showing that this sub graph is obtained by two different periods and five kemp chains, $K_1 = \{c_{12}, c_{11}, c_6, c_5\}$, $K_2 = \{c_{10}, c_7, c_4\}$, $K_3 = \{c_9, c_2, c_1\}$, $K_4 = \{c_8\}$, $K_5 = \{c_3\}$. A kemp swap K_2 and K_3 produces a new assignment by moving $\{c_9, c_7, c_{10}\}$ to p_j and $\{c_4, c_2, c_1\}$ to p_i as shown in (Figure 1.4b).

In this kemp swap one of the chains can be empty. Let us suppose that this empty kempchain is $K_6 = \{\}$. In this case kemp swap move generates into a single kemp chain interchange. It means that we have to move p_i with $(p_i \_ K) \cup (p_j \cap (K)$ and p_j with $(p_j \_ K) \cup (p_j \cap (K)$ where K is a non empty kemp chain. For example in Figure 1.4a, if we take the non empty chain K_1 then it produces an assignment by moving $\{c_{12}, c_{11}\}$ to p_j and $\{c_6, c_5\}$ to p_i. One can notice that this double kemp chain is the generalization of single kemp chain interchange.



(a) (b)

$$K_1 = \{c_{12}, c_{11}, c_6, c_5\},\ K_2 = \{c_{10}, c_7, c_4\},\ K_3 = \{c_9, c_2, c_1\},\ K_4 = \{c_8\},\ K_5 = \{c_3\}$$

Figure 1.4: Kemp chain illustration

This list of neighborhoods was used by many authors [114, 24] in literature, which are used to define genetic operators for genetic algorithm and also for constructing local searches for the sake of quality improvement of the solution. We have also used some of them for our local search method.

- N1: Select two lessons at random and swap periods.
- N2: The neighbourhood defined by an operator that moves one lesson from a period to a different one.
- N3: the neighbourhood defined by an operator, a lesson can be moved only if the corresponding period is empty.
- N4: Select two periods at random and simply swap all the lessons in one period with all the lessons in the other period.
- N5: Move the highest penalty lesson from a random 10% selection of the lessons to a random feasible period.
- N6: Carry out the same process as in N5 but with 20% of the lessons.
- N7: Move the highest penalty lesson from a random 10% selection of the lessons to a new feasible period that can generate the lowest penalty cost.
- N8: Carry out the same process as in N7 but with 20% of the lessons.
- N9: Select two periods based on the maximum enrolled lessons, say t_i and t_j. Select the most conflicted lesson in t_i and t_j and then swap them.
- N10: the neighbourhood defined by an operator that swaps the periods of two lessons
- N11: the neighbourhood defined by an operator that permutes three lessons in three distinct periods in one of the two possible ways other than the existing permutation of the three lessons.

One can also define particular swap moves as time swap, room swap depending on the representation of the solution.

1.7 Graph colouring and timetabling problem

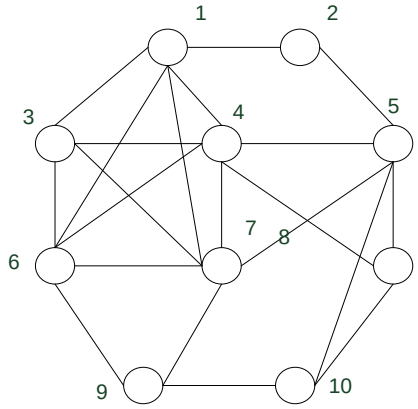
Timetabling problem has a resemblance with graph colouring problem. This was the basic reason that many early techniques used in timetabling algorithms were derived directly from graph colouring-based heuristics. One can see some of these from the review of Carter [129]. Now we explain how a timetabling problem can be solved with graph colouring. Given a simple and undirected graph *G* having *n* set of vertices, where V= $\{v_1, v_2, ..., v_n\}$ is a set of *n* vertices and *E* is the set of edges which joins different pairs of vertices of set *V*. This is a NP hard problem [159] which finds an assignment of colours for every vertex in *V* such that (i) vertices with common edge can not be assigned same color (ii) target is to find a solution

which uses the minimum number of colours. It is easy to convert the simplest timetabling problems to graph colouring problems (and vice-versa) by considering vertices as events and edge between any pair of vertices shows that there is a conflict between these lessons. So edge between any pair of vertices cannot have same colour. Then each colour in Graph colouring represents a period available for timetabling problem. Now task with respect to graph colouring problem is to use maximum colours which should be less or equal to the available periods for timetabling problem [20, 21]. Construction of timetable by graph colouring is shown in Figure 1.5.

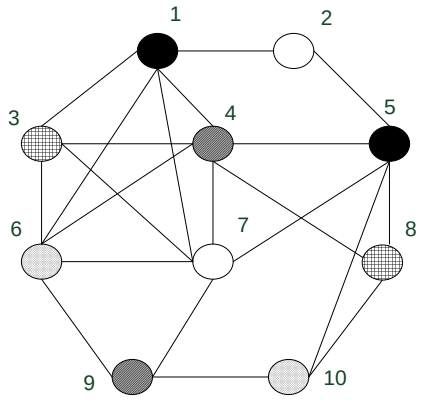
The term "chromatic number" number (commonly denoted χ) is used in graph colouring problems to refer the minimum number of colours which are required to colour a particular problem feasibly. Obviously for simple timetabling problem this means that the minimum number of required periods to schedule a clash free timetable for a particular problem.

The Clique is the second parallel that can be drawn between these two problems, which involves the identification of the features. A clique is a collection of vertices which are mutually adjacent, for example vertices 1, 3, 4, 6, and 7 in Figure 1.5 (b); this is a clique of size 5. It is worth noting that real world problems which uses graph colouring often have fairly large cliques. In many real world problems there are many events which are required not to be scheduled in the same period. In graph coulouring the vertices representing these events make a clique and it is trivial that no two vertices in this graph colorling could be assigned the same colour (or equivalently all events of a clique should be assigned different periods). So it can be deduced easily that for any graph colouring or timetabling problem if the maximum size of a clique is C then to make a feasible (that is $\chi \geq C$) timetable minimum C colours are required [159].

As mentioned earlier, graph coluring problem only exists when one is considering hard conflicting constraints. When other type of constraints such as room type constraints, ordering constraints etc. are added in the problem then it will add extra complications. However, regardless of this nearly all timetabling problems are featuring this graph colouring problem in some form or another form in their work. Many timetabling problem algorithms are still using various bits of heuristic information extracted from this graph colouring problems as a driving force to achieve their solutions [102].



(a) Given a simple timetabling problem having 10 lessons to schedule by using minimum number of periods, first problem is converted into its graph colouring equivalent.



(b) A solution is then found, which uses five colours (optimal number of colours) for this problem.

1	2	3	4	5
Event 1	Event 9	Event 6	Event 3	Event 2
Event 5	Event 4	Event 10	Event 8	Event 7

(c) The graph colouring solution is converted back into a valid timetable, where each colour represents a period.

Figure 1.5: Describing the relationship between the graph colouring problem and a simple timetabling problem (only considering lesson conflict constraint).

However graph couloring provides us more clues of many other lower bounds. One can notice that, feasible solution could not be found by using less than colours $\lceil n/m \rceil$, where *n* is total number of events and *m* is the total number of rooms. If number of colours is less than this number then room double booked constraint will be violated or some lessons will not be scheduled. One can also easily deduce that if $n > m \times p$ (where p is the total number of periods allowed for problem) then problem can not be solved feasibly because this means that number of events are more than the total number of places to schedule them. There are some other information about educational timetabling like if some lessons required a room of type *b* which is only one and the number of lessons which are requiring this room are Q then this timetabling problem could not be solved feasibly (if room type is used as a hard constraint) if $|Q| > p$.

The component of trivial course timetabling is given in [2]. Three courses are given Juggling, Math 101 and Algo 101. Juggling has one lesson; Math101 has four (Math101_1, Math101_2, Math101_3 and Math101_4) lessons and Algo101has three lessons (Algo101_1, Algo101_2, Algo101_3) to schedule. Math 101, Algo 101 have some common students and Jugling 101, Algo 101 also have some common students, means that they are making two classes. Now in associated conflict graph edge between two lessons showing that these two lessons can not take place concurrently. The dashed edge represents that requirement is coming from the enrollment of students in both courses (Figure 1.6). Presence of conflict constraint (lesson clash constraint) makes this timetabling problem similar to the graph coloring problem. However other issues like each course should be assigned in required type room, room capacity and many other ordering constraints can cause problem. Now many feasible colorings of a timetable might still not corresponds to a feasible timetables due to the addition of these extra constraints.

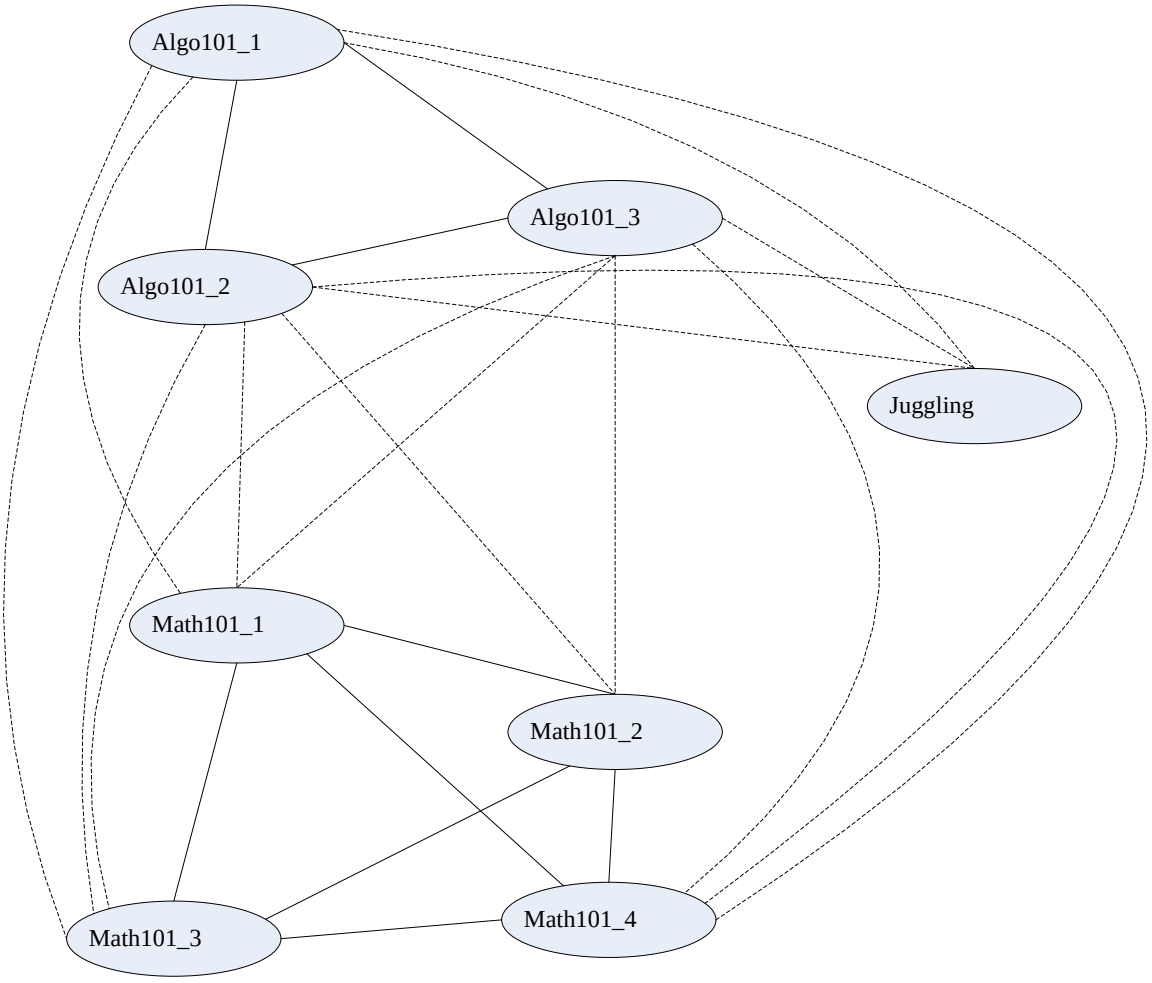


Figure 1.6: Associated conflict graph of lessons

1.8 Resolution techniques

Two techniques had been used in the literature to solve timetabling problems, namely exact methods and heuristic methods. In this section we shall present literature review on these two methods.

1.8.1 Exact methods

Exact methods consist of formulating and finding solution with integer programming models. Exact methods may not work to find optimal solution especially in short computing times for large size problem. This makes the reason to develop heuristics to get good solutions in moderate limit of time. We have presented generalized formulations for exam and course timetabling problems so we shall discuss integer programming techniques used in literature.

1.8.1.1 Integer programming techniques

Linear integer programming techniques have also been used to solve educational timetabling problems such as the algorithm of Carter [135], Tripathy [134] in the 1980s and Breslaw [120] provided a solution for the faculty assignment problem, a problem closely related to the timetabling problem, using linear programming models in the 1970s. The same problem was studied by McClure and Wells [139] and a solution was attempted again with the help of mathematical programming. Hultberg and Cardoso [155] formulated teacher assignment problem as a mixed integer programming problem and solved as a special case of the fixed charge transportation problem.

Among the first approaches in mathematical programming, in [156] the teacher assignment problem is combined with a form of the timetabling problem and solved through commercial software for goal programming. In a similar manner, Gosselin and Truchon [157] provided a linear programming formulation for the classroom allocation problem, a sub-problem of the university timetabling. Integer programming formulations for the school and the university timetabling problems as optimization problems are also discussed by Werra in [118], where the NP-completeness of both problems is shown even for simple versions. The author, however, chooses graph theory approaches for the solution of the problems under consideration. Extensions to this work, especially with regards to the so-called conflict matrix are presented by Tripathy in [158]. Given the difficulties of those days to solve large integer programming problems, the Lagrangean relaxation was proposed as a possible solution approach for the resulting model.

Many other articles have also used these techniques. Schimmelpfeng and Helber [121] described an integer programming approach which has been implemented at the School of Economics and Management at Hannover University, Germany, to create the complete timetable of all courses for a term and formulation was solved with CPLEX solver. Broek et al. [136] have divided the timetabling problem into four sub problems which are formulated as an integer linear programming problem. The goals of the four divided sub problems are specific as: maximize the number of assigned courses with urgency, maximize the total assigned workload, minimize the shortage of students to reach the minimum number of students of a section, optimize the timetable. The models introduced were solved by IP solver CPLEX.10.0. Boland et al. [27] applied blocking strategy, in which the classes can be

partitioned into sets of classes (or blocks) that will be timetabled in parallel. The problem of constituting the blocks and populating the classes is known as the course blocking and population problem. This formulation was made for high school timetabling and model was implemented in the modeling language AMPL, and solved using the ILOG package CPLEX 8.0.

Birbas et al. [122] presented a 0–1 integer programming model for the timetabling problem of Greek high schools. In their model, a binary variable indicates whether or not a specific lesson to be taught by a given teacher is to be held at a specific time of the week. The model generates timetables that satisfy all the hard and soft constraining. Werra [118] presented some basic models for course timetabling problem and these were described with an emphasis on graph theoretical models. Mingozzi et al. [123] presented a 0-1 linear programming formulation that requires an exponential number of variables corresponding to all feasible subsets of activities that can be simultaneously executed. Hassani [72] has developed an integer programming approach and solved the timetabling problem for Shahrood University of Technology, Iran. This approach was implemented on the problem of scheduling of 8 terms of graduation students. Two terms mean that the course which every student would take in one year. In order to solve the above problem, he formulated it with AIMMS using some identifiers.

Daskalaki et al. [12] presented integer programming formulation for university timetabling problem. The timetable for the Electrical and Computer Engineering Department in the University of Patras was used as a case study. Yakoob and Sherali [137,138] addressed a faculty-class assignment problem, periods for classes are initially assumed to be given and then integer programming model is constructed to minimize the individual and collective dissatisfaction of faculty members. Their second article's main focus is to design efficient class offering patterns while taking into consideration newly imposed gender policies. The both models were solved with CPLEX. Daskalaki and Birbas [24] considered the institutions which provide three to five year study program and provide a quite well structured curricula for their students. In this article author presents a two stage algorithm to solve this kind of problems. In the first stage they relax some constraints and solve them in second stage. They are relaxing courses compactness (if lessons of same course are scheduled on the same day they should be adjacent to each other) constraint at the first stage because in their opinion

these constraints are computationally heavier than the others. These constraints are recorded in the second stage and sub problems, one for each day of the week are solved for local optima.

Daskalaki et al. presented [28] an integer programming where problem was solved with modeling process result to solvable and flexible models. The flexibility was obtained by the multi-dimensional variables which allow details of the educational system to be modeled as constraints of the integer programming model. A variety of constraints could be represented by using this model. In order to check the capabilities of the proposed model for solving timetabling problem, the department of Electrical and Computer Engineering at the University of Patras was chosen. This is one of the largest departments in the Engineering School and has a five year program for students. During the first three years the students follow a general education program in Electrical and Computer Engineering and in the last two years each student join one of the four divisions of the department for specialization.

Natashia Boland et al. [27] discussed the solution which uses the procedure of blocking. Blocking is a set of classes which can be partitioned into subsets in such a way that all classes of a block can be scheduled in parallel. Each such type of subset is called block. Their method will induce a partition of the sessions available for the timetable in such a way that none of these sessions can be used for classes in other blocks. This means that each session will use only their block classes. This avoids the clashes to occur within blocks, and in this way it removes the clash conflict constraints. This further divide problem into two sub problems: the class blocking and population problem (CBPP) and the block timetabling problem (BTP). Class blocking and population problem means to make block of the given problem and block timetabling problem schedule the all blocks.

1.8.2 Metaheuristic methods

In this section we shall discuss about different metaheuristic techniques to solve educational timetabling problems. As we are going to propose two population based algorithms so we shall talk in detail about these algorithms. We have given a detailed review of evolutionary algorithms and Honey bee algorithms. We shall give a brief review of other techniques, which we found in literature like ant colony optimization, tabu search, simulated annealing and local search.

We shall also discuss some other techniques such as neural networks, hyper heuristics, these techniques can be considered as metaheuristic techniques. There are many good surveys on educational timetabling problems in the literature in the past including survey conducted by Carter and Laporte [129, 130] and Burke *et al*. [131, 73,132]. Other timetabling surveys can be found at [133, 34, 10], by Lewis and by Qu et al. [97,128].

1.8.2.1 Local search based algorithms

Metaheuristic methods are often classified as Local search based techniques and Population based techniques. Local search methods solve problems by searching from an incumbent solution to its neighbourhood. Different local search techniques can be distinguished by neighbourhood structures and moving operators within the search space. The search is guided by defined cost function, which is used to measure the quality of the generated solutions. The performance and efficiency depend a lot upon the parameters and search space properties. In this section, we shall talk about different local search based techniques and their variants used in literature to solve educational timetabling problem. We first start from tabu search algorithm.

1.8.2.1.1 Tabu search algorithms

Abdullah et al. [38] combined their genetic algorithm with sequential combined local search. In first step they find initial solutions for population pool by using these three heuristics: large degree heuristic, local search and tabu search. In this way they constructed feasible timetables for their population. Then results were improved by using genetic algorithm, in this algorithm single point crossover was used. After cross over and mutation they used local search heuristic to improve offspring.

Zhipeng Lu and Jin Kao Hao [1] present a tabu search algorithm about curriculum based course timetabling problem which was the part of International timetabling completion 2007. Their algorithm has three stages namely initialization, intensification and diversification. In first phase they generate feasible initial solution and to achieve this task they use greedy heuristic starting from an empty timetable. During this process at each time one adequate lesson is inserted into the timetable. At each step two distinct operations were done, first is to choose an unassigned lesson and second is to find appropriate place (room, period) for this

lesson. For lesson selection they use greedy colouring heuristics and when lesson has been chosen they select period for this lesson by using this heuristic.

They select period which is least likely to be used by other unassigned lessons on the next steps. They have used two neighborhood namely simple swap and kamp swap. They start tabu search algorithm by simple swap and when they get solution best local optima, they again start tabu search algorithm with other neighbourhood when using local optimum solution as an initial solution. Their intensification phase minimizes the soft constraints without breaking hard constraints any more. Their diversification criteria based on local search provide such mechanism which guides the search to escape from the current local optimum and move the solution towards new promising solutions.

In this article [87], a Tabu-based memetic algorithm is proposed by Salwani Abdullahand and Hamza Turabieh which hybridises a genetic algorithm with a Tabu Search algorithm as an improved algorithm for university timetabling problems. This algorithm uses a set of neighbourhood structures during the search process to obtain improvements in solution quality. They propose a sequence of neighbourhood structures to understand its effect on the search space. They evaluate random, best and general sequences of neighbourhood structures in this work. Those neighbourhood structures are penalized which do not generate better solutions. In this article Batenburg and Palenstijn [30] used evolutionary and tabu search algorithm in parallel. They show that for large problems using a parallel variant of tabu search with genetic algorithm shows significant improvements in the obtained timetable comparatively to the memetic algorithm. But this procedure will enhance the run times.

1.8.2.1.2 Local search algorithms

Duong Tuan Anh et al. [4] proposed a two stage algorithm, in which first stage will get feasible solution not having any hard constraint and in second stage they try to minimize soft constraints. In first they use backtracking free constructive method with local search and look ahead. The local search used in first stage is Minimum Conflict Hill Climbing with dynamic constraint weighing mechanism. For improving the initial feasible solution produced at first stage was improved in second stage by using two candidate local search methods. Hill climbing algorithm is a first candidate method where local minima was escaped by using stochastic strategy. Hill climbing algorithm also is a second candidate method where local minima was escaped by using short term memory strategy.

Edmund Burke [2] wrote this article about partial solutions achievements when trying to get a good time table where task is to minimize soft constraints. This article covers an approach of such problems which could be a multiphase exploitation of multi objective sub models. In this problem at first step only one difficult component of problem and related objective function is taken. This would find a partial solution which would further define interesting neighbourhood in the search space of the entire problem. Variable aggregation can be performed by picking the initial component at the first stage and by exploring neighbourhoods at next stage to ensure feasible solutions. Then it is easy to use integer programming to implement heuristics to produce solutions with bounds on their quality. This study was performed on International timetabling competition track whose datasets comes from Udine university (Italy).They use objective restricted neighbourhood generator in their heuristic for the assignments of periods to lessons, with minimizing the violations of two period related soft constraints.

Patrick De Causmaecker [3] represents a decomposed heuristic for solving a university course timetabling problem. The datasets were taken from KaHo Saint-Lieven School of Engineering. They use the technique to make pillars of similar lessons to make problem less complex. In this way no doubt search space will become smaller than original search space. Then this problem was solved by sequentially evaluating the constraints one by one by using the procedure that solution obtain in one stage will be used in the next stage as an initial solution. In fact soft constraints were minimized one by one and by using this procedure they get good results for their problem compared to solve all constraints at once. Ersoy [56] proposed a combination of hill-climbing and memetic algorithms as a hyper-heuristic framework. They tested its performance on the Carter benchmark datasets. The results showed that a memetic algorithm based hyper-heuristic using a single hill climber at a time gave the best results among other variants of hill climbing hyper-heuristics proposed by the same authors.

Qu and Burke [90] constructed a unified graph-based meta-heuristic (GHH) framework, upon which a number of local search-based algorithms as the high level heuristics are studied to search upon sequences of low-level graph coloring heuristics. To gain an in-depth understanding on their framework, they addressed some fundamental issues concerning

neighborhood structures and characteristics of the two search spaces, the search spaces of the heuristics and the actual solutions. Furthermore, they investigated efficient hybridizations in GHH with local search methods and mentioned issues concerning the exploration of the high-level search and the exploitation ability of the local search.

Jacques A. Ferland and Alain Lavoie [26] also used a heuristic iterative procedure where assignment of one event can be modified at each iteration. This is an exchange procedure, which applies first to find a feasible solution (satisfying hard constraints) and then it improves the objective function value. A geometrical interpretation of the exchange is given to understand the theoretical framework of the procedure. Two more procedures are introduced to avoid jumping outside the feasible domain or at local optimum. The first procedure relies on Lagrangean relaxation and second one uses inductively more than one exchange per iteration.

1.8.2.1.3 Simulated annealing algorithms

Simulated Annealing algorithm was initially proposed by Kirkpatrick et al. [140] for solving combinatorial optimization problems. Simulated annealing procedure is prevented by accepting deteriorating moves that worsen the objective function value. In this way search procedure can be protected by getting trapped with a local optimum. This algorithm application in timetabling problem can be found in works by Abramson [143] and Bullnheimer [142]. A review of Simulated Annealing algorithms for solving the University Course Timetabling Problem can be found in Kostuch [18]. Abramson [143], Melicio and Caldeira [144] and Elmohamed *et al*. [145] proposed simulated annealing algorithms. Elmohamed *et al.* [145] considered the timetabling problem of Syracuse University in the USA and used a weighted-sum scoring function which penalized violations of the hard constraints. They also used general simulated annealing practices to enhance the algorithm performance.

Merlot *et al.* [154] proposed a three stage approach for solving examination timetabling which consisted of constraint programming, simulated annealing and then hill climbing.
Constraint programming was used to construct a feasible timetable. The other two stages were used to minimize the soft constraints through the use of simulate annealing and a hillclimber using Kempe chain interchanges. The algorithm was tested by using real-world problem datasets from the University of Melbourne and also for various benchmark problem datasets

such as the Carter datasets. Defu Zhang et al. [146] presented a simulated annealing based approach which used new extended neighborhood structure obtained by performing a series of swaps of pairs of assignments during two periods. This algorithm was tested on high school timetabling problems.

Sara Ceschia et al. [147] also presented a simulated annealing approach for a post enrolment course timetabling problem. Their solution was based on a relatively simple single-step algorithm. They used preprocessing, constraint reformulation for the improvement of the local search and room assignment procedure based on attractiveness which allowed them to refrain from using the matching algorithm. Their single-step procedure considers the soft constraints from the very beginning and it can save computational time.

Aldy Gunawan [148] proposed which used Lagrangian relaxation and a simulated annealing algorithm. They used Lagrangian relaxation approach to generate initial feasible solutions in the construction phase. Simulated annealing algorithm was used for the improvement of initial feasible solutions. Thompson and Dowsland [31] proposed a two phase simulated annealing algorithm. The first phase finds a feasible solution and second phase satisfies the secondary objectives and soft constraints. They discuss in detail about the parameter used to control the algorithm and how to model a simulated annealing framework for successful implantations by using judicious choices in both areas. They tested different neighborhoods and cooling schedules over a data from various institutions. They concluded that neighborhoods based on the graph-theoretic concept of Kempe chains are the most efficient regardless of the objectives or size of the problem.

1.8.2.1.4 Great deluge algorithms

Turabieh et al. [60] proposed a great deluge algorithm for university course timetabling. This depends upon attraction–repulsion movement for solutions in the search space. Algorithm starts with a population of randomly generated feasible initial solutions and an attraction–repulsion mechanism is used to check the quality of the great deluge algorithm. Then a great deluge algorithm is applied to increase the solution quality. This algorithm's performance was also tested on Socha datasets. Fukushima Makoto [91] proposed a hybrid algorithm which includes Simulated Annealing, Local Search and Great Deluge to perform for solving the university course timetabling problem. The Great Deluge algorithm (GD) is a generic algorithm applied to optimization problems. It is similar in many ways to the hill-climbing and simulated annealing algorithms. The name comes from the analogy that in a great deluge

a person climbing a hill will try to move in any direction that does not get his/her feet wet in the hope of finding a way up as the water level rises.

In a typical implementation of the GD, the algorithm starts with a poor approximation, S, of the optimum solution. A numerical value called the badness is computed based on S and measures how undesirable the initial approximation is. The higher the value of badness the more undesirable the approximate solution is. Another numerical value called the tolerance is calculated based on a number of factors, often including the initial badness.

A new approximate solution S', called a neighbour of S, is calculated based on S. The badness of S', b', is computed and compared with the tolerance. If b' is better than tolerance, then the algorithm is recursively restarted with S: = S', and tolerance: = decay (tolerance) where decay is a function that lowers the tolerance (representing a rise in water levels). If b' is worse than tolerance, a different neighbour S* of S is chosen and the process repeated. If all the neighbours of S produce approximate solutions beyond tolerance, then the algorithm is terminated and S is put forward as the best approximate solution obtained [92]. In [91] they proposed two enhancements in their algorithm, where one of three soft constraints is satisfied when obtaining a feasible timetabling solution and a few parameters for escaping local minima are introduced in the Great Deluge algorithm which is used for reducing the violations of the soft constraints.

Shaker and Abdullah [39] presented curriculum based course timetabling (CB-CTT) problem and they solved it by a hybridization of great deluge algorithm with kempe chain neighbourhood structure. The problem was solved in two steps, in first step they used a graph-based heuristic to construct a feasible timetable and in second step they improve their feasible solution by employing a hybrid approach. The algorithm was tested on the curriculum-based course timetabling problems as described in the 2nd International Timetabling Competition (ITC2007).

1.8.2.2 Population based algorithms

Many population based algorithms have given good performance in educational timetabling problems. In this section, we discuss different population based techniques and their variants.

1.8.2.2.1 Ant colony algorithms

Socha et al. [149, 51] used the ant colony optimisation metaheuristic to university course timetabling problem. They proposed two ant-based algorithms, an Ant Colony System and a MAX-MIN system. They gave a qualitative comparison between them. Every ant first constructs a complete assignment of events to periods using heuristics and pheromone information, at each step in both of the algorithms. Then local search procedure was used for further improvements of timetables. The major differences between the two approaches are the way that heuristic and pheromone information is interpreted, and the approaches used for updating the pheromone matrix. However MAX-MIN system generally gave better results when experiments were conducted on a range of problem instances.

Clemens Nothegger et al. [151] proposed a new approach to solve the problem of Post Enrolment Course Timetabling by using Ant Colony Optimization. In this algorithm ants successively construct solutions based on pheromones (stig- mergy) and local information. The main feature of this algorithm was the use of two distinct but simplified pheromone matrices for improving performance but still provide enough edibility for effectively guiding the solution construction process. Furthermore a local improvement method was embedded and algorithm was applied to datasets used for the First and Second International Timetabling Competition (ITC 2002 and ITC2007). Michael Eley Max-Min [152] presented ant colony algorithm and used ANTCOL approach. They compared the performance of this algorithm with other approaches presented in the literature and with modified graph coloring algorithms. He tested his algorithm on examination timetabling problem.

Michael Eley [54] proposed two more ant colony algorithms which were applied simultaneously to construct and improve examination timetables. The first algorithm was (MMASET) based on the MAX–MIN Ant System. This algorithm was used by [51] for course timetabling problems. The second algorithm was (ANTCOL-ET) a modified version of ANTCOL. This was originally used by [55] to solve graph coloring problems. Dowsland and Thompson [150] proposed Ant colony optimization algorithm for the examination scheduling problem. It built on an existing implementation for the graph colouring problem to make non conflicting timetables and went on to consider the introduction of a number of additional practical constraints and objectives. They modified and improved the original algorithm. They tested the performance of proposed algorithm on carter bench mark datasets.

1.8.2.2.2 Partical swarm optimization algorithm

Der-Fang Shiau [93] proposed a new meta-heuristic algorithm that is based on the principles of particle swarm optimization for course scheduling problem by designing an 'absolute position value representation for the particle, allowing instructors that they are willing to lesson based on flexible preferences, the maximum number of teaching-free periods and the lecturing format and employing a repair process for all infeasible timetables. Furthermore, a local search mechanism is incorporated into the proposed PSO in order to explore a better solution improvement.

Ioannis and Beligiannis [65] have proposed hybrid particle swarm optimization (PSO) based algorithm for high school timetabling problems. This algorithm has produced feasible and efficient high school timetables. They have used real world data coming from many different Greek high schools for their experimental work. The performance of this algorithm is better than existing approaches applied to the same school timetabling input data while using the same evaluation criteria.

Qarouni-Fard et al. [113] have proposed the particle swarm optimization algorithm to the classic timetabling problem. Their algorithm is inspired by approaches belonging to the evolutionary paradigm where involved metaheuristic is tweaked to suit for the problems such as timetabling, graph coloring or bin packing. For evolutionary algorithms, it means to substitute the "traditional operators" with newly defined operators that evolve fit groups rather than fit items. Authors have applied a similar idea for their PSO algorithm and performance of proposed algorithm is better in comparison with previously used approaches.

1.8.2.2.3 Genetic algorithms

Evolutionary algorithms have been extensively and successfully used to solve timetabling problems since their first applications to timetabling problems at the beginning of the 1990s.

Now we explain some main stages of evolutionary algorithms which are performed during the execution of these algorithms.

1.8.2.2.3.1 Initialization

Evolutionary algorithms have been extensively and successfully used to solve timetabling problems since their first applications to timetabling problems at the beginning of the 1990s.

Genetic algorithms are used generally with an initial population. This initial population sometimes is generated randomly and sometimes by using special techniques to make a higher quality initial solutions for population [129, 59, 115, 53]. After getting the initial solution repairing can be mainly done on violation of hard constraints. We have shown in Figure. 1.7, 1.8 that how these repair functions work in evolutionary algorithms.

/*A course comes more than once in a day*/

Repeat

Find the course S1 occurring more than once in a day D1;

Case 1: Try o replace to free timeslot in a day where S1 not occurs;

If Teacher (S1) not free in the free timeslot

Repair Teacher's conflict and replace;

Case 2: Find a day where S1 not occurs;

Swap with any timeslot of S2, provided S2 not in D1;

Until (Repairing all courses's conflict)

Figure 1.7: Repair procedure for Course's conflict

Different heuristics were also used in literature to make initial solutions for population. Normally these were graph colouring heuristics [100, 102, 116, 83], again their use depends on the nature of the problem and solution representation used by the solver. Now we shall give a glimpse of these heuristics as follows.

/* Case.1. Swapping Free timeslot with Theory Timeslot of

same day*/

Repeat

Find timeslots TS1<- free; TS2 <- Theory; TS3<- Theory in a day Di of Class Ci

To swap (TS1, TS2);

If Teacher (TS2) not free in TS1

{

Swap (TS1,TS3);

Swap (TS2,TS1);

}

Else if

Swap (TS1,TS2);

End if;

Until (Repairing all Teachers' Conflict);

/*Case.2. Swapping free timeslot with Theory Timeslot of same day with intern adjustment*/

Repeat

Find TS1<-free; TS2<- Theory in a day Di of Class Ci

To swap (TS1, TS2);

If Teacher (TS2) not free in TS1

{

Make free Teacher (TS2) in TS1 by adjusting in the class

Ci+1/ Ci+2;

Swap (TS1, TS2);

}

Until (Repairing all Teachers' Conflict);

/*Case. 3 . Swapping free timeslot with Theory Time slot of some other day*/

Repeat

Find TS1<-free; TS2<- Theory in a day Di of Class Ci

To swap (TS1, TS2);

If Teacher (TS2) free in TS1

{

If (! subject's conflict)

Swap (TS1, TS2);

Else

{

Normalize Course's conflict;

Swap (TS1, TS2);

}

}

Until(Repairing all Teacher's Conflict);

Figure 1.8: Repair procedure for teacher's conflict

(1) Courses are first sorted in descending order of their complexities, and then periods and rooms are allotted to them, now it depends on the solver how he defines the complexity.

(2) Choose the unscheduled course with the smallest number of possible room-period pairs to which it can be feasibly assigned in the current timetable.
(3) Choose an unscheduled course randomly.
(4) Choose the period that the least number of other unscheduled courses could be feasibly assigned to this period in the current timetable.
(5) Choose the place that defines the period with the fewest events in.
(6) Choose a room period pair randomly.
(7)Those courses have priority which has small number of available periods and a large number of unassigned lessons. This heuristic is akin to the greedy colouring heuristic [116].

Once a lesson of a course is chosen then for period selection this criterion is used, select a period among all available ones that is least likely to be used by other unfinished courses at later steps [1, 83]. We have given a procedure of such type heuristic to form initial solutions for population Figure 1.9.

Build (*tt, U*).
1. **If** (len (*tt*) < *t*)
2. Open (*t* – len (*tt*)) new timeslots;
3. **Insert-Events** (*tt*, *U*, 1, len (*tt*));

Insert-Events (*tt, U, l, r*).
1. **While** (there are events in *U* with feasible places in *tt* between timeslots *l* and *r*)
2. Choose an event *e* from *U* that has feasible places in *tt*;
3. Pick a feasible place *p* for *e*;
4. Move *e* to *p*;
5. **if** (*U* = ϕ) **end**;
6. **else**
7. Open $\lceil |U| / m \rceil$ new timeslots;
8. **Insert-Events** (*tt*, *U*, *r*, len (*tt*));

Figure 1.9: Build procedure Lewis thesis [102]

1.8.2.2.3.2 Selection

The most used selection operators are described as follows.

(1) *Elitism*

This method is used to select the best chromosome or a few best chromosomes for the next population. This procedure can increase algorithm performance very quickly because it preserves the best found solutions [112].

(2) *Roulette Wheel Selection*

This procedure is a probability based method where better chromosomes have the more chances of selection. It could be understood by a simple example, consider a roulette wheel where all chromosomes are placed in the population. Every chromosome has its place as big as fitness function value is good, it is shown in the following picture [110] Figure 1.10.

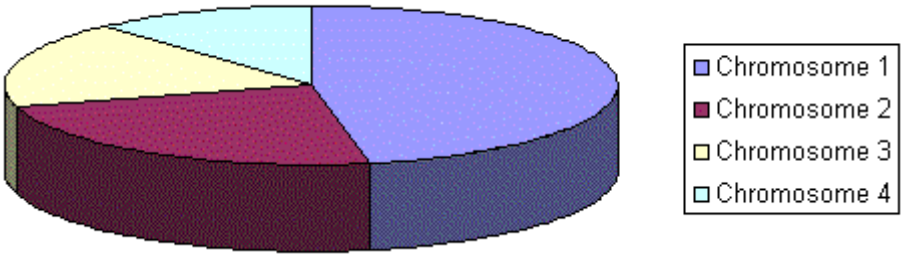


Figure 1.10: Picture expressing roulette wheel selection

Then to select a chromosome a dice is thrown. The chromosome with best fitness function value has more chance of selection than not good enough value chromosome.

(3) *Tournament Selection*

A specific number of chromosomes are selected from the population, and the one having the best fitness value will go for the next generation [111, 52].

1.8.2.2.3.3 Cross over

In the following we will now describe the different crossover operators used in our experiments.

(1) *One-point crossover*

First of all a single crossover point on both parents organism strings is selected. Then all data beyond that point is swapped in either organism string between the two parent organisms. The obtained organisms are the new children (as shown in Figure.1.11).

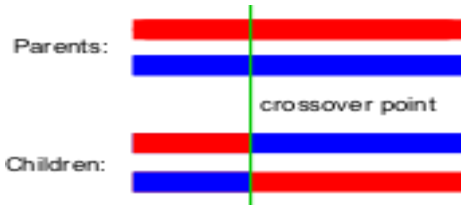


Figure 1.11: Picture expressing one point crossover

(2) *Two-point crossover*

This is the same as one point crossover, here one selects two points on the parent organism strings and everything between the two points is swapped between the parent organisms and it makes two children (as shown in Figure.1.12). Similarly one can define N-point crossover.

Figure 1.12: Picture expressing two point crossover

(3)*Uniform Crossover*

This Crossover uses a fixed mixing ratio between two parents. This operator enables the parent chromosomes to contribute on gene level instead of segment level.

If one considers the fixed mixing ratio is 0.5 then offspring gets approximately half of the genes from first parent and the other half from second parent. Cross over points can be randomly chosen as shown below Figure. 1.13.

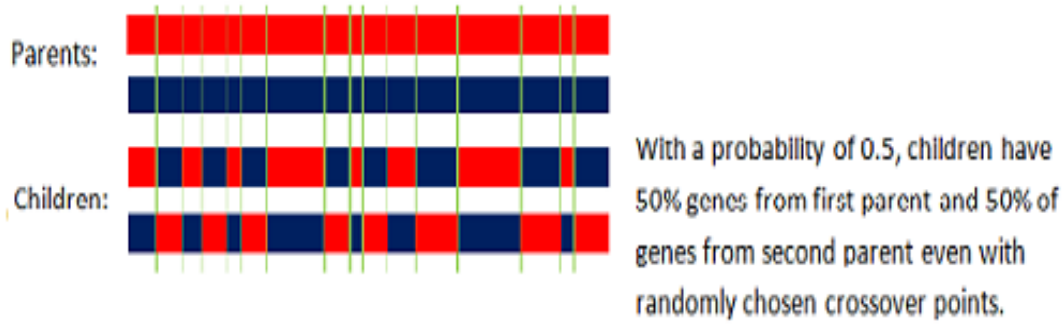


Figure 1.13: Picture expressing uniform crossover

The Uniform Crossover checks each gene in the parent strings for exchange with a probability of 0.5. So each gene of both parents has equal chance of selection. The evidence suggests that it is a more exploratory approach to crossover than the traditional approaches that maintains longer schemata [108,111, 103].

N-point crossover is quite different from the Uniform crossover. N-point crossover exchanges material between points of parents chromosome strings but in uniform crossover each parents gene has a probability of changing, for example if probability is 0.5 it means that both parents have equal chance of giving gene to child chromosome. After cross over, mutation is normally performed.

1.8.2.2.3.4 Reproduction

Reproduction is the procedure of making new children to bring new population. There are generally two types of reproduction: generational reproduction and steady-state reproduction.

(1) *Generational reproduction*

The whole of a population is replaced at each generation in generational reproduction. The procedure which is often used is N/2 times. This means, produce two offspring from selected two parents (according to the selection procedure) and finally produce N new chromosomes where N is the population size [75, 76, 77].

(2) *Steady-state reproduction*

This method selects two parents according to the selection procedure, generates one or two children and installs the result back into that population; the least fit of the population is destroyed. These off springs are the only generated solution for this generation [70, 24, 83].

1.8.2.2.3.5 Genetic algorithms in educational timetabling

Sadaf Naseem Jat and Shengxiang Yang [70] proposed a guided search genetic algorithm, i.e., GSGA for solving the university course timetabling problem. This is a steady state genetic algorithm where a guided search strategy and a local search technique are integrated; Chiarandini et al. [71] used local search method for their algorithm as well. Steady state genetic algorithm means that only one child solution is generated per iteration/generation. Useful information are stored in guided search strategy. These information contained room, period pairs of those lessons for which there is no hard or soft constraints violation. These information are used to guide the generation of child into the next population. Local search is used to improve the quality of individuals by using three neighborhood structures.
Bratkovic [75] used a genetic algorithm to solve laboratory exercises timetabling problem. This algorithm used especially crafted crossover and mutation operators that are closed over the space of feasible solutions and these genetic operators are designed to satisfy only hard constraints. A local search was used to reduce the soft constraints and it improves the solution quality.

Olivia Rossi-Doria and Ben Paechter [83] described a steady state evolutionary algorithm with binary tournament selection and random mutation. One point crossover is used to make offspring. This algorithm evolves the choice of the heuristics to be used at each step of the building process of a timetable. This idea was originally published in Blum et al. [84].

This genetic algorithm proposed by Hitoshi Kanoh and Yuusuke Sakamoto [86] uses installed knowledge base and an infection operation. The knowledge base means set of candidate partial solutions of the final solution which are built from past years timetables and about request of teachers related to their work choices. So in this way current timetable can preserve the advantages of past timetables. This algorithm performance is tested on the timetables of University of Tsukuba.

Amin Jula and Narjes Khatoon Naseri [89] presented a new hybridized genetic algorithm by adding local search algorithm. They applied Cerebellar model of Articulation Controller (CMAC) as a trainable machine to calculate and update mutation rate related to different possible states in the process of algorithm execution. CMAC is a neural network that simulates the structure and functions of a part of the brain called Cerebellum. CMAC model is based on associative memory cells and searching table. Their experimental results obtained showed that heuristics applied in this approach lead to better solutions with CMAC.

Dilip Datta et al. [94] presented the problem of preparation of class timetable in IIT Kanpur, India. The entire timetable is composed of two phases. The first phase contains all the common compulsory classes of the institute, which are scheduled by a central team. The second phase contains the individual departmental classes. The evolutionary algorithms have been exploited in this article to schedule the classes. They showed that using NSGA-II-UCTO, a multi-objective EA-based university class timetable optimizer, a number of trade-off solutions, in terms of multiple objectives of the problem, could be obtained.

The significance of crossover while finding global optimal in maintaining diversity has been identified in M.Nandhini [95] and found that combination of grade selection with combinatorial Partially Matched Cross over of better performance. Sadaf Naseem Jat and Shengxiang Yang [70] state a guided search algorithm for the university course timetabling problem, where guided search strategy and local search is integrated to solve the datasets. The guided search strategy uses a data structure to store full information about events. This data structure (memory) does not store whole timetables but it stores some best scheduled events from the timetable. Data structure is regularly maintained after few generations. This data structure can be formed by selecting N set of best individuals from the whole population. Then take one individual solution to check its all events that whether its penalty is zero or not, if its penalty is zero then all information about this event like room and period will be added in the data structure. Then next individual solution will be picked and all information about each zero penalty event will be added in the data structure. Then these data structure information are used to guide the generation of children for the next population. Then author uses a local search technique to improve the individual solutions searching through their neighborhood structures in the solution space.

Adaptive genetic algorithms are GAs in which the parameters like population size, crossover probability or mutation probability can be varied while the GA is in progress. A simple example can be the following, the mutation rate is not fixed in the population but it changes according to the performance of the algorithm. If the fitness of the population does not improve for long time, the mutation rate increases. A lower mutation rate is chosen again if the population shows an improvement as described in Zafer Bingul [107].

It is important to use appropriate values for parameters such as mutation rate, crossover rate and population size to get good performance of genetic algorithms [103]. So selection of these parameters has a wide range to get best solutions [104]. So some authors used the technique of varied based parameters (these parameters were adjusted during the run) for genetic algorithms which varied according to the performance of the genetic algorithm [105,106]. It is observed that solver takes low mutation rate, high crossover rate at the beginning of the algorithm and high mutation rate and low crossover rate towards the end of the run. The reason behind is that at the beginning they want to go towards good solutions and take mutation rate less. But when research goes towards end, solution gets stuck and they increase mutation rate for getting diversity. The relationship between population features and GA parameters are very complex and non-linear [107].

Burke, E.K [57] proposed a variant of variable neighbourhood search for solving examination timetabling problems. Genetic algorithm was used to select a subset of neighborhoods. They tested the performance of algorithm on the Carter benchmark problem datasets. Pillay and Banzhaf [52] presented a two-stage informed genetic algorithm for examination timetabling problems. First phase got feasible solutions and second stage was used to eliminate soft constraint violations. In both phases genetic algorithm was used to construct solution and then to improve it.

Landa-Silva and Obit [58] proposed an evolutionary non-linear great deluge algorithm for course timetabling problem. Tournament selection was used as a selection operator. Better improved individual was replaced with worst individual in population. Algorithm performance was tested on the Socha datasets and results showed that the hybridization between the non-linear great deluge and evolutionary operators produced good quality results. This algorithm is similar to the one given by Abdullah [59].

1.8.2.2.3.6 A simple example of genetic algorithm

We take a simple example to show how basic components of genetic algorithm work through optimization of a function. The function is defined as $f(x) = x \sin(1/x)$. The objective is to find a minimum for this function on the interval [0, 0.5], i.e. to find a value x_0 such that $f(x_0) \leq f(x)\ \forall x \in [0,0.5]$. The first derivative of this function is as follows.

$$f'(x) = \sin(1/x) - \frac{\cos(1/x)}{x} = 0$$

We can find the maximum or the minimum values of a function by equating its first derivative to zero.

We can write its first derivative in the following form as well.

$f'(x) = x\tan(1/x) = 1$ And it shows that there are infinite many solutions. However we want to solve it with genetic algorithm.

Representation

A chromosome is a binary vector and represents a real number. This binary vector has 32-bits since the processor allows this number. This means that the length of the domain of variable *x*, which is 0.5 in this example, is divided $2^{32} - 1$ equal size ranges. So a binary string can be converted to a real number as follows:

First we convert binary string of length 32 from base 2 to base 10.

$$x^* = ((b_{31} b_{30} \ldots b_0))_2 = \left(\sum_{i=0}^{31} b_i 2^i \right)_{10}$$

This gives the corresponding real number.

$$x = x^* \frac{1}{2^{32} - 1}\ 0.5$$

where 0.5 is the length of the domain.

For example, a chromosome (11110011110010101001011110100111) is representing the number 0.476155032 in [0, 0.5]

since $x^* = (11110011110010101001011110100111)_2 = (4090140583)_{10}$

and the real number is $x = (4090140583)\ \frac{1}{2^{32} - 1}\ 0.5 = 0.476155032$.

It can be easily noticed that (00000000000000000000000000000000) and (11111111111111111111111111111111) represent the boundaries of the domain, 0 and 0.5 respectively.

Initial population

The initial population consists of 10 random chromosomes in form of binary vectors each of 32 bits. Following are the 10 chromosomes c_i where $1 \leq i \leq 10$ and their corresponding real value x_i on the interval [0, 0.5].

c_1 = (01101110010010110001111111101001) x_1 = 0.21541691

c_2 = (11000000011001001110100110111111) x_2 = 0.37538495

c_3 = (11110011110010101001011110100111) x_3 = 0.47615503

c_4 = (10001100010110101011110101110100) x_4 = 0.27412979

c_5 = (10010100101100110001000010100000) x_5 = 0.29042866

c_6 = (10000110110110000100100000000000) x_6 = 0.26336884

c_7 = (01100101000110110101101011110000) x_7 = 0.19747433

c_8 = (11010110010111110111001010110111) x_8 = 0.41869696

c_9 = (01011001000010001110001110010111) x_9 = 0.17389594

c_{10} = (11001010100101001111100010101010) x_{10} = 0.39566781

Objective function

Here objective function *obj for* chromosomes is equal to the original function *f*, *obj(c) = f(x)* where *x* represents the corresponding real value and *c* represents the chromosomes.

Objective function has an important role as it gives information about good solutions in terms of fitness. These are fitness values of the chromosomes in our population:

obj (c_1) = f(x_1) = -0.214885912

obj (c_2) = f(x_2) = 0.172565550

obj (c_3) = f(x_3) = 0.410983902

obj (c_4) = f(x_4) = -0.132941257

obj (c_5) = f(x_5) = -0.086269622

obj (c_6) = f(x_6) = -0.160509512

obj (c_7) = f(x_7) = -0.185396112

obj (c_8) = f(x_8) = 0.286388252

obj (c_9) = f(x_9) = -0.088302975

obj (c_{10}) = f(x_{10}) = 0.228031778

We can see that the fittest chromosome in the population is chromosome c_1 because it gives the smallest value and c_3 is the worst chromosome with the highest fitness value.

Genetic operators

There are two classical operators in genetic algorithm namely crossover and mutation. One point crossover is applied on two fittest chromosomes c_1 and c_7. We choose a random number, say 21 for crossover point. This means that we split the chromosomes at the 21st gene and then we exchange the parts of the chromosomes as given below:

c_1 = (**01101110010010110001**1 | 11111101001)

c_7 = (01100101000110110101 | **01011110000**)

We get these two children from parents:

o_1 = (**01101110010010110001101011110000**)

o_2 = (01100101000110110101111111101001)

Now the fitness of these new chromosomes and their corresponding real values can be calculated as following.

$f(o_1) = f($0.21541676$) = -0.214885813$

$f(o_2) = f($0.19747604$) = -0.185400708$

The first child o_1 has a better fitness value than its parents however this improvement can be seen till ninth decimal point.

The next step is to mutate the children to get better chromosomes. For this purpose, we select 6 random genes and change their values. The bit value '1' is changed to '0' or vice versa.

An example of this type of mutations is given below:

o_1 = (01101110010010110001101011110000)

o_{1m} = (011**<u>100</u>**10**<u>1</u>**1001011000110101111**<u>11</u>**0)

o_2 = (01100101000110110101111111101001)

o_{2m} = (011**<u>110</u>**01**<u>1</u>**00110110101111111101**<u>11</u>**1)

After applying mutation, we re-calculate the fitness of these new chromosomes and the corresponding real values below:

$f(o_{1m}) = f($0.22420582$) = -0.217113164$

$f(o_{2m}) = f($0.23751354$) = -0.208197846$

It is easy to notice that objective function value of mutated children is better than their parents also from children which are not mutated. These mutated children act as parents for the next generation, this means that each iteration starts with a better population. Thus genetic

algorithm finds optimal or near optimal solutions gradually by this way. In this example, we have explained only one iteration.

1.8.2.2.4 Memetic algorithms

Population based algorithms have been used to solve educational timetabling problems in the literature. Cote et al. [53] proposed an evolutionary algorithm to minimize the timetable length and the conflicting examinations as much as possible. They used two local searches (tabu search and variable neighbourhood descent) instead of recombination operators to deal with hard and soft constraints violations. However replacement of recombination operators with two local searches increased number of parameters that should be tuned.
Ersoy et al. [56] proposed a combination of hill-climbing and memetic algorithms as a hyper-heuristic framework. They tested its performance on the Carter benchmark datasets. The results showed that a memetic algorithm based hyper-heuristic using a single hill climber at a time gave the best results among other variants of hill climbing hyper-heuristics proposed by the same authors.

Salwani Abdullah et al. [59] employed evolutionary algorithm together with local search. The main technique used in this evolutionary algorithm is a light mutation operator followed by a randomised iterative improvement algorithm. The crossover operator is not used in this approach. The local search approach discussed in Abdullah et al. [15] is used as the operator and is applied after the mutation takes place; similar work was also done by Soolmaz Massoodian and Afsaneh Esteki [74].

Ender Özcan and Alpay Alkan [76] solved a University Exam Preparation School Timetabling Problem (PSTP), this is a course timetabling problem. A memetic algorithm is used as an incremental multistage approach. At each stage a new subset of unscheduled events is selected using some criteria. Within a population of candidate solutions unscheduled events for the selected subset of events are randomly generated. This population is exposed to the memetic operators and whenever some termination criteria are satisfied, then another new subset of events is chosen. This procedure is repeated until all unscheduled events are scheduled and some additional termination criteria are satisfied. The size of individuals incrementally increased as the new subsets of events is added for optimization at each stage. In this way, no portion of the search landscape is ignored.

Olivia Rossi-Doria and Ben Paechter [77] proposed a memetic algorithm to solve university course timetabling problem, which uses the representation and local search, due to Socha and Chiarandini [78, 79].This algorithm is a Steady-State evolution algorithm, proposed by Withley [81], where only one offspring solution is produced from two parents at each generation. Local search is used in two phases, where first phase makes an infeasible timetable feasible by reducing the number of periods used and second phase is used to improve the quality of a feasible timetable by minimizing the number of soft constraint violations. This algorithm uses constructive heuristics as used in [82, 80] to make initial solutions for initial population.

Ender Ozcan et al. [85] proposed a hybrid method, ‘‘Interleaved Constructive Memetic Algorithm’’ (ICMA) that interleaves memetic algorithms with constructive methods. This algorithm starts working by using an active subset of all the lessons. In multiple construction stages ICMA increases the active sub sets one by one and eventually include all of them starting with single set. At each stage a memetic algorithm is used to improve the quality of current partial solution before the next construction stage. This algorithm is used to solve Preparation School Timetabling Problem (PSTP).

Ricardo Santiago-Mozos et al. [88] proposed a two-phase heuristic evolutionary algorithm to construct personalizing timetables. They tackle the problem of assigning a feasible and personalized timetable for every student. Students are allowed to choose a set of priority courses and non-priority courses. Priority courses will always be assigned. This algorithm uses two-phase heuristic with evolutionary algorithm to solve course timetabling problem. First phase of heuristic is used for the assignment of priority courses and when this task is done then the second heuristic manages the non-priority course’s assignment. This algorithm is used to make personalized timetables of the School of Telecommunications Engineers, Universidade de Vigo (Galicia, Spain).

1.8.2.2.5 Honey bee algorithms

We have proposed a honey bee algorithm for educational timetabling problem. In this section we present some details about the algorithm. There are two major types of Honey bee

algorithms. First type is a honey bee colony algorithm which was mostly used to solve educational timetabling problems [45, 46, 47]. The honey bees colony Algorithm was introduced in 2005 by Pham et al. [66,67]. Honey bee colony algorithm is about the food collection of the bees. It is a nature inspired algorithm [61, 62, 63, 64, 67].

Define M, E(t), and S(t) to be the spermatheca size and the queen's energy and speed at time t respectively
Initialize the queen's genotype at random
select a worker at random, apply it to improve the queen's genotype, and update its fitness
while the stopping criteria are not satisfied
t = 0, generate a drone at random
initialize E(t) and S(t) randomly and the energy reduction step ° to 0:5£E(t)
M
while E(t) > 0
evaluate the drone's genotype
if the drone passes the probabilistic condition, **and** the queen's spermatheca is not full, **then**
add the drone's sperm to the queen's spermatheca
t = t + 1; E(t) = E(t - 1) - γ ; S(t) = 0:9* S(t - 1)
with a probability of S(t) flip each bit in the drone's genotype
end while
for brood = 1 to total number of broods
select a queen in proportion to her fitness and a sperm from that queen at random
generate a brood by crossovering the queen's genome with the selected sperm
mutate the generated brood's genotype
use a worker selected in proportion to its fitness to improve the drone's genotype
update the worker's fitness based on the amount of drone's improvement
end for
if the best brood is better than the queen **then** replace the queen with the best brood
kill all broods
end while

Figure 1.14: Original HBMO for SAT [68]

The second type is honey bee mating optimization (HBMO) algorithm [48]. This was proposed by Abbass in 2001 [68, 69]. This naturally inspired algorithm simulates the process of real honeybees mating. It was successfully used for solving job shop scheduling, data

mining, 3-sat, integrated partitioning/scheduling, stochastic dynamic programming and continuous optimization, nonlinear constrained and unconstrained optimization problems [49]. We have given pseudo code of original honey bee mating algorithm in Figure.1.14.

The strength of this algorithm is to explore simultaneously and exploit problem search space. This is achieved by the queen's transition in the search space and employing a local search at each iteration. These features of honey bee algorithms make it different from other population based algorithms that had been used on educational timetabling problems [50, 51, 52]. Honey bees are selected to create the population of the initial hive. Previous work [96, 97, 98, 99, 100] showed that in many cases, random generation methods may not necessarily guarantee a good quality in some cases. Therefore, in this work, we employ heuristics which are described in later chapter (Chapter 7) to generate an initial population of drones.

1.8.2.2.5.1 Review of Honey bee algorithms

Alzaqebah and Abdullah's [45] article speaks about how to use artificial bee colony for solving the examination timetabling problems. This algorithm works with three categories of bees namely employed, onlooker and scout bees. Employed bees fly around the search space to find food source and come back in hive to share collected information with onlooker bees. Onlooker bees relay on these information and they choose their food source according to these information. While scout bees are those employed bees whose food source has been abandoned. Now these scout bees start to search a new food source randomly without any information. If they find new source of food where amount is more than the previous one in their memory they memorize the new source and forget the previous one. This algorithm employed local and global search methods simultaneously where local search method are used through employed bees and onlooker bees. While global search methods are carried out by onlooker bees and scout bees. The purpose of this combination is to have a balance between exploration and exploitation process.

Nguyen et al. [46] presented a hybrid algorithm which combines honey bees algorithm and harmony search algorithm. The part of harmony search algorithm handles intensification and diversification. Diversification is handled by pitch adjustment and random selection which helps to retain good local search solutions. Random selection explores the search space more widely while pitch adjustment makes the new solution good enough to the existing good solutions. The intensification in Harmony search algorithm is controlled by memory

consideration; this leads the search process toward the searching space of good solutions. This algorithm has two phases of optimization. First phase use honey bees algorithm using neighborhood search and random search while second phase uses harmony search using memory consideration, random selection and pitch adjustment.

Alzaqebah and Abdullah's algorithm [47] starts with initial solution obtained from graph coloring heuristics and size of population is equal to the number of scout bees. Each scout bee evaluates the solution according to objective function. Highly ranked solutions are selected for local exploration by other bees (foragers) that are directed to the neighbourhood of the selected solutions by the scout bees. Then for each selected solution the number of foragers is allocated by this rule. If scout bee returns from one of the best solutions, performs the "waggle dance" this means that it recruits some specific amount of mates for local exploration. The scout bees that visit the elite solutions among the best sites recruit specific foragers for a neighbourhood search. The scout bees that visit the remaining solutions recruit a group of foragers for a neighbourhood search. Algorithm gives more tries for the elite solutions because elite solutions are the most promising solutions in the search space.

Reference	Problem description	Classes, Courses, Teachers, Rooms, Periods	Algorithm	Results
[44]	ITC-2007 (Examination timetabling problem)	Benchmark datasets	No	Problem mathematical formulation is proposed
[45]	ITC-2007 (Examination timetabling problem)	Benchmark datasets	Bee colony algorithm	Tested on real world datasets
[48]	Carter et al. bench mark instances [82] and Socha et al. bench mark instances [51]	Benchmark datasets	Honey bee mating algorithm	Tested on real world datasets
[57]	Carter et al. bench mark instances	Benchmark datasets	Variable neighbourhood search	Tested on real world datasets
[164]	Universiti Kebangsaan Malaysia	Datasets are not given	Timetabling Software	Results are not given
[161]	University in Thailand	758, 128, 111, no, 60	Stochastic optimisation	Tested on real world datasets
[167]	University of Valencia, Spain	93, 84, 200, 24, 65	Tabu search	Tested on real world datasets
[13]	Universit´a del Sannio, Benevento, Italy	14, 64, 48, 12,50	Cutting plane algorithm	Tested on real world datasets
[72]	Shahrood University of Technology, Iran	8,75, 25, 3, 38	Integer programming	Tested on real world dataset
[27]	Xavier College, Melbourne	90,32, 68, blocks 6, no	Integer linear programming	Real data modified

Reference	Problem description	Classes, Courses, Teachers, Rooms, Periods	Algorithm	Results
[1]	ITC-2007 (Curriculum based course timetabling problem)	Benchmark datasets	Tabu search	Tested on modified real world datasets
[2]	ITC-2007 (Curriculum based course timetabling problem)	Benchmark datasets	Hybrid metaheuristic	Tested on modified real world datasets
[3]	KaHo Sint-Lieven School of Engineering	212, 1215 lessons, 133, 67, 95	Local search based metaheuristic	Tested on real world datasets
[65]	Greece high school datasets	11, 385 lessons, 35, no, 35 (11 datasets of approximately this size)	Particle swarm optimization	Tested on real world datasets
[87]	ITC-2007 (Curriculum based course timetabling problem)	Benchmark datasets	Tabu based memetic algorithm	Tested on modified real world datasets
[93]	Kun-Shan University Taiwan	8,6 course for each class ,17, 10 ,40 (small size data)		
[147]	ITC-2007 (Post enrolment course timetabling problem)	Benchmark datasets	Simulated annealing	Generated datasets
[148]	University in Indonesia	Datasets are not given	Simulated annealing	Generated datasets+ Real datasets
[151]	ITC-2007 (Post enrolment course timetabling problem)	Benchmark datasets	Ant colony optimization	Generated datasets
[160, 168]	Statistics department of Hacettepe University	4, 36, 27, 6, 40	Tabu search	Tested on real world datasets

Table 1.1: Lists papers related to course and examination timetabling problem

1.8.3 Brief review of RCPSP

Francisco Ballestin et al. [124] study the case when pre-emption is allowed for processing jobs. The generalized case of this problem is m_PRCPSP, which means that one job can be pre-empted at most m times but case studied in this paper is 1_PRCPSP, for reason, it is easy and also if one pre-empted in more time, objective function normally does not improve.

Sonke Hartmann and Dirk Briskorn [125] give an overview over extensions of the RCPSP such as multiple modes, minimal and maximal time lags. The extensions are classified according to the structure of the RCPSP. They summarize generalizations of the activity concept, of the precedence relations, of the resource constraints and discuss the notations, models and classification schemes. Sonke Hartmann and Rainer Kolisch [126] first present a literature survey. They discuss about X-pass method, in which they use SGS (Serial schedule generation scheme). Here they present the results and find out the performance of many state of the heuristics on some benchmark datasets. They compare the results and point out the most performing procedure.

Ana Viana and Jorge Pinho [9] used multiobjective metaheuristics to solve the resource constrained project scheduling problem (RCPSP). They applied multiobjective versions of simulated annealing and tabu search inorder to minimise the makespan, lateness of activities and the violation of resource constraints. They checked the performance of these metaheuristics on randomly generated instances. The benefit of this multiobjective version is that it is near to real world problems where many objectives are required to achieve.

Job shop scheduling problem is a special case of resource-constrained project scheduling problem. Peter Brucker et al. [22] presented a classification scheme compatible with machine scheduling. Project scheduling researchers used a variety of symbols to denote one and the same subject. So there is a gap of notations and classification scheme between machine scheduling and project scheduling and also a lot of articles are publishing which makes this problem more worst. Authors tried to minimize this gap and presented a classification scheme (description of the resource environment, the activity characteristics, the objective function etc.), compatible with machine scheduling. They also reviewed some recent developments in exact and heuristic algorithms for the single-mode and the multi-mode RCPSP.

Krzysztof Fleszar and Khalil Hindi [23] proposed a solution method based on variable neighbourhood search. They coded solution by using activity sequences which are available in form of precedence constraints. These sequences became valid active schedules through a serial scheduler. The solution is coded by using activity sequences that are valid in terms of precedence constraints. The sequences are turned into valid active schedules through a serial scheduler. Two types of move strategies were used to explore solution space by generating valid sequences, effective lower bounding and precedence augmentation were employed to reduce the solution space. Rainer Kolisch and Sonke Hartmann [29] discussed heuristics for solving resource-constrained project scheduling problem (RCPSP). This was an update of previous review. They summarized and categorized a large number of heuristics recently proposed in literature. They evaluated these heuristics in a computational study and compared them on the basis of their standardized experimental design and on the basis of these results they discussed features of good heuristics.

1.9 Conclusion

This chapter basically consists of introduction of educational timetabling problems and the approaches used to solve these problems. We have discussed different types of timetabling problems and especially our main concerned is educational timetabling problems. Timetabling problems related to educational institutions have discussed in detail with different types of constraints which make this type of problem really difficult. A lot of solution approaches used graph colouring heuristics to solve these problems, so we have discussed graph colouring problems in comparison with educational timetabling problems. Many metaheuristics have been used in literature to solve timetabling problems like local search based and population based metaheuristics. Our major focus is also metaheuristics because we have proposed metaheuristics to solve timetabling problems. Local search based metaheuristics need neighbourhood structures for its functioning so we have discussed these neighbourhood structures used in literature in detail.

A major part of this chapter covers different solution techniques used in literature. It is also worth mentioning that each technique has its own advantages and disadvantages, we can not decide that one technique is universally superior to any other one. Instead we can say that certain techniques are more appropriate to certain kind of problem-situations and certain types of user requirements. Thus it looks reasonable, when selecting a specific approach for one's

own timetabling problem to keep in mind that produced solution will be used ultimately by real people. So it is required by algorithm to be fast, reliable and can produce good solution according to the desires of the users.

We have noticed that there are mainly two types of solution techniques. First type of technique is the one which solves problem directly without giving any focus on hard or soft constraints but this procedure tries only to minimize objective value of the function where hard constraints have more penalty value than soft constraints. Second type of technique is the one which solves the problem in two steps. In the first step the algorithm solves hard constraints of the problem and in the second step the algorithm tries to solve soft constraints while maintaining the solution feasible (means no violation of hard constraints). We shall say that this is not a concrete classification.

We have noticed that many algorithms have been used to solve a particular problem or the author's own institution problem. So this is quite understandable that obviously they will be motivated to solve that specific problem. But according to research point of view it is difficult to judge how their algorithms performance is good with respect to others's algorithms. As university timetabling problems often do not have any standardized problem definitions and do not have many different problem instance libraries available for benchmarking algorithms. Thus there is a need of more and generalized problem instance libraries to analyze and compare different algorithms performance.

We have tried to present details of the domain in our Chapter 1. But if someone does not have prerequisite knowledge of these problems, we feel that this chapter is not sufficient for his foundation in this domain, one can study referred literature given in our general introduction.

Chapter 2

2 TRANSFORMATION OF COURSE TIMETABLING PROBLEM TO RCPSP

In this chapter, we have proposed two equivalent mathematical formulations which transform course timetabling problem to RCPSP (resource constrained project-scheduling problem). In Section 2.1, we have given a brief introduction to RCPSP. In Section 2.2, we have discussed why we choose single mode RCPSP for transformation and benefits of this transformation. In Section 2.3, we have presented our mathematical models and chapter is concluded in Section 2.4.

2.1 The resource constrained project scheduling problem (RCPSP)

The classical resource constrained project-scheduling problem (RCPSP) may be stated as follows. A project consists of a set of n activities numbered 1 to $\bar{j}$, where each activity has to be processed without interruption to complete the project. We consider additional activities *j* = 1 and *j* = $\bar{j}$ representing the single source and single sink activity of the network respectively. The duration of an activity *j* is denoted by d_j, where d_1=0 and $d_{\bar{j}}$ = 0. There are *R* renewable resource types. The availability of each resource type *r* in each time period *t* is a_{rt} units, *r* = 1,…,*R*. Each activity j requires u_{jr} units of resource *r* during each period of its duration where u_{1r} = 0, $u_{\bar{j}r}$=0, *r* = 1,…, *R*. All parameters are assumed to be non-negative integer valued. There are precedence relations of the finish-start type with a zero parameter value (i.e., FS = 0) defined between the activities. In other words, activity *i* precedes activity *j* if *j* cannot start until i has been completed. The structure of a project can be represented by an activity-on-node network G = (V,A), where V is the set of activities and A is the set of precedence relationships. F_j (P_j) is the set of successors (predecessors) of activity *j*. It is

assumed that $1 \in P_j$, $j = 2,\ldots, \bar{j}$ and $\bar{j} \in F_j$ $j = 1,\ldots, \bar{j}$-1. The objective of the RCPSP is to find a schedule *S* of the activities, i.e., a set of starting times ($s_1,...,s_{\bar{j}}$), where $s_1 = 0$ and the precedence and resource constraints are satisfied, such that the schedule duration *T*(*S*) = $s_{\bar{j}}$ is minimized.

Our problem is based on non preemptive activities but in RCPSP activities could be preempted during processing at integer points in time, i.e., the fixed integer processing time d_j of activity *j* may be split into *j* = 1, 2 , …, d_j process units. Time windows can be specified for every activity, [EF*j*, LF*j*], which denote the earliest and latest finishing time for activity *j*, and [ES*j*, LS*j*] which denote the earliest and latest starting time for activity *j*. This problem specifies a minimal and maximal time lag between tasks. A minimal time lag specifies that an activity can only start or finish when the predecessor activity has already started (finished) for a certain time period. A schedule *S is* called feasible if in each time period *t* the total resource demand is less than or equal to the availability a_{rt} of each resource type *r*, and the given precedence constraints are fulfilled. We call a problem of finding a feasible schedule with completion times *Cj* such that $Cj \leq T$ for $j = 2,\ldots, \bar{j}$-1 a search problem or feasibility problem. A search problem with threshold value *T* has a solution if and only if a schedule *S* exists such that the makespan $C\max = \max_{j=2}^{\bar{j}-1}(C_j) = \max_{j=2}^{\bar{j}-1}(s_j + d_j)$ is not greater than *T*. The RCPSP is usually formulated as the problem of finding a feasible schedule which minimizes the makespan. Other important objective functions besides are based on cost functions $f_j(t)$ for the activities. One has to find a feasible schedule which minimizes the total costs $\sum_{j=2}^{\bar{j}-1} f_j(C_j)$ [141].

2.2 Basic single-mode RCPSP and course timetabling

Our purpose is to transform educational timetabling problem (course timetabling) to resource constrained project scheduling problem. There are six major classes of RCPSP: 1. Basic Single-Mode RCPSP 2. Basic Multi-Mode RCPSP 3. RCPSP problems with non regular objective functions 4. Stochastic RCPSP 5. Bin-packing-related RCPSP problems 6. Multi-resource-constrained project scheduling problems.

If a resource constrained project scheduling problem (RCPSP) uses a single execution mode for every activity, with specific time and resource requirements is called single mode resource constrained project scheduling problem. Here we shall only discuss and use single mode RCPSP because in timetabling problem, courses are assigned to teachers in first stage and total duration for each course is also predefined. So by using single mode RCPSP, scheduling of these courses could be done. In our formulations we use set of lessons instead of set of courses to transform problem in RCPSP for that purpose we decompose firstly the courses durations into set of lessons.

One benefit to transform timetabling problem in RCPSP is that durations of lessons can be set according to choice but normally solvers take lesson length uniform for the easiness and the other thing is that if there are precedence constraints (i.e lesson i of duration d_i must be taught before lesson j of duration d_j) between lessons then this kind of formulations will be more beneficial to use than others timetabling formulations. The other aspect of attention is a new dimension of thought and beauty of mathematical work which can open new rooms for researchers. We are thinking on the idea that how many other features of RCPSP and its generalizations can be attached to timetabling problems that these problems could be solved by using RCPSP solvers or techniques.

2.3 Transformation of timetabling problem to RCPSP

This section defines many sets and sub sets, which are used to formulate mathematical models. Our two proposed formulations are the part of this section as well.

2.3.1 General features of the models

In this section we shall define our sets and sub sets, which will be used in our formulations.

- A set of lessons $J = \{1,\ldots, \bar{j}\}$.
- A set of types of rooms $R = \{1,\ldots,\bar{r}\}$.
- A set of rooms $Y = \{1,\ldots,\bar{y}\}$.
- A set of periods $T = \{1,\ldots, \bar{t}\}$. T is a set of periods, which all have same length.
- A set of classes $C = \{1,\ldots, \bar{c}\}$. Class is a set of lessons which have common students.

- A set of teachers $P = \{1,\ldots, \bar{p}\}$. Each lesson will have a teacher previously assigned to it.

Some additional parameters and sets are defined on the basis of previous sets to make easy the presentation of model.

a_r = Constant room availability of room type *r*

a_{rt} = Availability of rooms of type r in period *t*

u_{jr} = Use of room type *r* per period by job *j*, which is always one

J_p =Set of lessons taught by teacher *p*

J_r = Set of lessons requiring rooms of type *r*

Y_r = Set of rooms of type *r*

ES_j = Earliest starting time of lesson *j*

LF_j = Latest finishing time of lesson *j*

P_j = Set of lessons which precede lesson *j*

F_j = Set of lessons which follow lesson *j*

d_j = Duration of an activity *j*

Lesson 1 and lesson $\bar{j}$ are dummy lessons, which are called generally source and sink. To ease presentation, durations and resource usages for these lessons is considered zero. Earliest starting and latest finishing times can be obtained by a forward and backward pass respectively. Starting with $ES_1 = EF_1 = 0$, the forward pass calculates earliest starting and finishing times as follows.

$ES_j = \max\{EF_i / i \in P_j\}$; $EF_j = ES_j + d_j$ for $j = 2,\ldots, \bar{j}$.

The backward pass is performed beginning with $LF_{\bar{j}} = LS_{\bar{j}} = \bar{t}$. This gives latest finishing and starting times LF_j and LS_j as follows.

$LF_j = \min\{LS_h / h \in F_j\}$; $LS_j = LF_j - d_j$ for $j = \bar{j}-1,...,1$

E(t) = $\{j /J$ and $ES_j + 1 \le t \le LF_j\}$. This is a set of lessons which are eligible to schedule for a period t.

The latest finishing and earliest starting times correspond to time points delimiting periods. So it is important to clear difference of time period and time point for better understanding of the formulation. Two time points t and t+1 define the start and the end of period t+1

respectively. If earliest starting time of any lesson is ES_j then the earliest time period for its execution could be $ES_j + 1$.

We have formulated timetabling problem in two different ways on the prototype of Resource constrained project scheduling problem. We have formulated high school timetabling problem in a two different ways.

2.3.2 Proposed mathematical model 1

With these variables, the first formulation can be defined as follows.

{ $x_{jt} = 1$ if lesson *j* is scheduled in period *t*, $x_{jt} = 0$ otherwise ;} for $j \in J$ and $t \in T$

Max $\sum_{t \in T} \sum_{j \in J} x_{jt} d_{jt}$

$$\sum_{t \in T} x_{jt} = d_j \qquad j \in J \tag{1}$$

The constraint (1) ensures that each lesson is scheduled for d_j periods.

$$d_j.(x_{jt} - x_{j,t+1}) - \sum_{q=ES_j+1}^{t} x_{jt} \le 0 \qquad j \in J,\ t \in [ES_j + 1, \ldots, LF_j - 1] \tag{2}$$

The constraint (2) is a non preemption constraint which ensures that processing of each lesson is not interrupted.

$$d_i.x_{jt} - \sum_{q=ES_i+1}^{t-1} x_{it} \le 0 \quad j \in J,\ i \in P_j,\ t \in [ES_j + 1, \ldots, LF_i] \tag{3}$$

The constraint (3) shows that a lesson *j* must not be started before all its predecessors have been processed completely.

$$\sum_{j \in E(t)} u_{jr}.x_{jt} \le a_{rt} \qquad r \in R,\ t \in T \tag{4}$$

The constraint (4) guarantees that the number of lessons scheduled in period *t* requiring rooms of type *r* will be less than or equal to the number of rooms of type *r* available at period *t.*

$$\sum_{j \in J_p} x_{jt} \le 1 \qquad t \in T,\ p \in P \tag{5}$$

The constraint (5) ensures that teacher *p* cannot teach more than one lesson at period t.

$$\sum_{j \in c} x_{jt} \le 1 \qquad t \in T,\ c \in C \tag{6}$$

The constraint (6) ensures that class *c* cannot attend more than one lesson at period t.

$x_{jt} \in \{0,1\} \qquad j \in J,\ t \in T$

Where d_{jt} is the desirability to schedule lesson *j* in period *t*, basically this is preference to teach lesson for teachers in periods because sometimes they are performing some other administration duties and some teaching periods are more suitable for them than others.
Objective function can be used according to demand, if one wants to schedule these lessons as early as possible, one can use Min $\sum_{t=ES_n+1}^{LF_j} t.x_{jt}$, which is same as one uses in RCPSP (minimize the project completion time).

2.3.3 Proposed mathematical model 2

The second formulation is proposed using these variables. This is equivalent to the first formulation.

{ x_{jyt} = 1 if lesson *j* is scheduled in period *t* at room *y*, x_{jyt} = 0 otherwise ;} for $j \in J$, $y \in Y$ and $t \in T$

$$\text{Max} \sum_{y \in Y} \sum_{t \in T} \sum_{j \in J} x_{jyt} d_{jt}$$

$$\sum_{y \in Y} \sum_{t \in T} x_{jyt} = d_j \qquad j \in J \tag{7}$$

$$\sum_{y \in Y} d_j.(x_{jyt} - x_{j,t+1}) - \sum_{y \in Y} \sum_{q=ES_j+1}^{t} x_{jyt} \le 0 \qquad j \in J,\ t \in [ES_j + 1, \ldots, LF_j - 1] \tag{8}$$

$$\sum_{y \in Y} d_i.x_{jyt} - \sum_{y \in Y} \sum_{q=ES_i+1}^{t-1} x_{iyt} \le 0 \qquad j \in J,\ i \in P_j,\ t \in [ES_j + 1, \ldots, LF_i] \tag{9}$$

$$\sum_{j \in J_r} \sum_{y \in Y_r} x_{jyt} \le a_{rt} \qquad r \in R,\ t \in T \tag{10}$$

$$\sum_{y \in Y} \sum_{j \in J_p} x_{jyt} \le 1 \qquad t \in T,\ p \in P \tag{11}$$

$$\sum_{y \in Y} \sum_{j \in c} x_{jyt} \le 1 \qquad t \in T,\ c \in C \tag{12}$$

$x_{jyt} \in \{0, 1\} \qquad y \in Y,\ j \in J,\ t \in T$

The constraints 1 and 7, 2 and 8, 3 and 9, 4 and 10, 5 and11, 6 and12 are representing the same constraints.

2.4 Conclusion

The Resource-constrained project scheduling problem (RCPSP) is concerned with single-item or small batch production where limited resources have to be allocated to dependent activities over time. Over the past few decades, a lot of work has been done with the use of optimal solution procedures for this basic problem type and its extensions. Brucker and Knust [166] had discussed how timetabling problems can be modelled as a RCPSP. Authors discuss high school timetabling and university course timetabling problem as an example. We have formulated two mathematical formulations of course timetabling problem which are the prototype of single-mode RCPSP [127].

These formulations are basically linear integer programming and could be solved by using linear programming solvers like CPLEX, LINGO etc. It would be interesting to correlate more features of timetabling problem to RCPSP. We expect, this effort would be thought provoking and would be a new addition in this domain. The purpose of the work is to show how course timetabling problem can be transformed into RCPSP.

Chapter 3

3 A GENERIC MODEL OF UNIVERSITY COURSE TIMETABLING PROBLEM

In this chapter, we have presented a new 0-1 linear integer programming formulation for university course timetabling problem. The mathematical model for the problem provides many operational rules and requirements which are needed in many institutions. We have formulated a generic model by gathering many constraints from different university environments in a single formulation. Remaining of the chapter is organized as follows.

In Section 3.1, we have explained the terms, sets, sub sets and parameters required to formulate the mathematical model. In Section 3.2, we have proposed the mathematical formulation of our generalized university course timetabling problem. In Section 3.3, we have demonstrated how the constraints of this model can be used as hard and soft and how one can make objective function using these soft constraints. In Section 3.4, we have explained the procedure of the construction of a timetable by using a small problem instance. Finally, the chapter is concluded in last Section 3.5.

3.1 General features of the proposed model

In this section, we are going to propose a generic mathematical formulation for university course timetabling problem which covers many constraints of different university environments.

3.1.1 Used entities of the model

Our generalized problem consists of the following entities.

Courses and Teachers

Each course has fixed number of lessons to be assigned in distinct periods. It is attended by given number of students and is taught by a teacher.

Days, Periods per day and Total Periods

We have a number of teaching days in a week. Each day has a fixed number of periods. Total periods are the product of number of periods per day and number of days.

Slots

A triplet of room-period-day is called one slot. Total number of slots can be found by multiplying total number of days, total number of rooms and total number of periods per day.

Rooms

Each room has a given capacity (number of available seats). Each room also has a type called room type. Two rooms of same capacity can have different types.

Class

A class is a group of courses which have common students. Thus the courses of one class must not be scheduled at the same period.

3.1.2 Notations, sets, sub sets and parameters

Now we shall define our sets and present in detail the problem requirements.

- A set of course $C = \{1,\ldots, \bar{c}\}$. Each course has a fixed number of hours per week, which can be taught during the week.
- A set of rooms $R = \{1,\ldots, \bar{r}\}$. Rooms can be of different types, including computer rooms and laboratories for specific subjects. Each room has a fixed capacity.
- A set of total periods $P = \{1,\ldots, \bar{p}\}$. P is a set of periods (all of the same length, 1 hour). Total number of periods $|P|$ can be found by multiplying number of periods per day with total number of days.
- A set of days $D = \{1,\ldots, \bar{d}\}$. D is the teaching days of the week.
- A set of classes $K = \{1,\ldots, \bar{k}\}$. K is a set of classes (classes are groups of students attending exactly the same courses).

- A set of teachers $T = \{1,\ldots, \bar{t}\}$.

Each course will have a teacher previously assigned to it.

- A set of types of rooms $X = \{1,\ldots, \bar{x}\}$.

n_c = *Number* of teaching hours to be scheduled per week for every course $c \in C$

$n_{c\min}$ = Minimum daily number of teaching hours (no less than $n_{c\min}$ teaching hours have to be assigned to day *d*, if the course *c* is scheduled in day *d*)

$n_{c\max}$ = Maximum daily number of teaching hours (no less than $n_{c\max}$ teaching hours have to be assigned to the day *d*, if the course *c* is scheduled in day *d*)

t_d = First period of the morning session in day *d*

l_d = First period of the afternoon session in day *d*

$C_k \subset C$ = Set of the courses that class *k* should attend, for every class *k*. Unlike the School timetabling problem, some classes can be joined to attend the same courses, i.e., $C_{k1} \cap C_{k2} \neq \phi$.

$C_t \subset C$ = Subset of courses taught by teacher *t* for every teacher *t*

e_t = Maximum weekly number of teaching days allowed for the teacher *t*

g_l = Maximum gap between two teaching hours (lessons) of any teacher *t*

$l_{\max}$ = Maximum daily number of teaching hours allowed for any class

$l_{k\max}$ = Maximum daily number of teaching hours allowed for every class $k \in K$

$l_{k\min}$ = Minimum daily number of teaching hours limit for every class $k \in K$

$\max l_{cd}$ = Maximum daily number of teaching hours allowed for every course $c \in C$

$\min l_{cd}$ = Minimum daily number of teaching hours allowed for every course $c \in C$

$\max_{lt}$ = Maximum load per day for every teacher *t*

$\min_{lt}$ = Minimum load per day for every teacher *t*

$\max_{lp}$ = Maximum lessons scheduled per period *p*

d_{cp} = Desirability of scheduling the course *c* at period *p*, usually d_{cp} measures the desirability of the period *p* for the teacher *t* ($c \in C_t$)

$P_r \subset P$ = Set of periods for which room *r* is available, for every $r \in R$

$P_t \subset P$ = Set of periods for which teacher *t* is available, for every $t \in T$

$C_x \subset C$ = Set of courses requiring rooms of type *x*, for every $x \in X$

$R_x \subset R$ = Set of rooms of type *x*, for every $x \in X$

m_{xp} = Number of rooms of type *x* available at period *p*, for every $p \in P$ and $x \in X$

$P_c \subset P$ = Set of pre assigned periods for course *c*, for every $c \in C$

$P_d \subset P$ = Set of periods for day *d*, *for* every $d \in D$

$P_{d''} \subset P$ = Set of last periods of days

$A \subset C$ = Set of pre assigned courses

$\bar{p}_c \subset P$ = Set of forbidden periods for course *c*, for every $c \in C$

$P_k \subset P$ = set of periods for which class *k* is unavailable, for every $k \in K$

$F \subset C$ = set of courses for which periods are forbidden (all periods are not available)

$R_f \subset R$ = set of rooms which should be free in timetable, it means that no course $c \in C$ should be scheduled in any period $p \in P$, in these rooms.

$R_t \subset R$ = set of rooms, which are in the preference list of teacher *t*, for every $t \in T$

$P_f \subset P$ = set of periods which should be free in timetable, it means that no course $c \in C$ should be scheduled in any room $r \in R$, in these periods.

$\min_{wc}$ = Minimum working days for course *c*

$\max_{wc}$ = Maximum working days for course *c*

s_c = Number of students in course *c*

a_r = capacity of room *r*

We have defined C_k, group of students attending exactly the same courses but it can also be defined another way, group courses in such a way that any pair of courses in the group have one or more common students. The other way is more general than the first way.

A timetable is an assignment of courses *C* to rooms *R* and to periods *P*. A timetable fulfills some basic requirements, which are normally called hard constraints, mostly common for every institution. It also satisfies some other constraints whose satisfaction is not mandatory but their fulfillment shows that how good timetable is.

3.1.3 Decision variables

We have defined these decision variables to formulate our mathematical model. We have defined four binary variables.

- x_{crp} = 1 if course $c \in C$ is scheduled in room $r \in R$ at period $p \in P$, x_{crp} = 0 otherwise;
- u_{cd} = 1 if course $c \in C$ is assigned to day $d \in D$, 0 otherwise;
- Ψ_{td} = 1 if $d \in D$ is a teaching day for teacher $t \in T$, 0 otherwise;
- z_{rc} = 1 if room $r \in R$ is used by course $c \in C$, 0 otherwise;

3.1.4 Objective function

We have proposed a generalized model for course timetabling. This generalized model consists of many constraints. For this generic model the objective function is also generic and can be adopted for specific problems. These constraints of the generic model can be used as hard or soft constraints in a specific problem. One can write the objective function when he knows exactly which constraint is soft and which constraint is required as hard. We have shown in Section 3.3, how one can write objective function from a specific problem by taking an example.

3.2 Integer programming formulation for generalized problem

In this section, we have presented our mathematical formulation. This formulation has many different types of constraints and we have divided them in six main categories, which are described in the following.

3.2.1 Hard constraints

In this section, we have written four constraints which are normally used as hard constraints in different university environments.

For every course $c \in C$, n_c hours a week must be scheduled.

$$\sum_{r \in R} \sum_{p \in P} x_{crp} = n_c \qquad c \in C \tag{1}$$

For every class $k \in K$, class *k* cannot attend more than one course at period $p \in P$.

$$\sum_{c \in C_k} \sum_{r \in R} x_{crp} \leq 1 \qquad k \in K,\ p \in P \tag{2}$$

For every teacher $t \in T$, teacher *t* cannot teach more than one course at time $p \in P$.

$$\sum_{c \in C_t} \sum_{r \in R} x_{crp} \leq 1 \qquad p \in P,\ t \in T \tag{3}$$

For every room $r \in$ R, room *r* cannot host more than one course at time $p \in P$.

$$\sum_{c \in C} x_{crp} \leq 1 \qquad p \in P, r \in R \tag{4}$$

3.2.2 Period related constraints

For every $p \in P$, the number of courses scheduled in period p requiring room type *x* will be less than or equal to the number of rooms of type *x* available at period *p*.

$$\sum_{r \in R_x} \sum_{c \in C_x} x_{crp} \leq m_{xp} \qquad p \in P, x \in X \tag{5}$$

For every $p \in P$, if course is not adequate to room type, it could not be assigned to that room.

$$\sum_{r \in R_x} \sum_{c \in C - C_x} x_{crp} = 0 \qquad p \in P, x \in X \tag{6}$$

For every period $p \in P_{d''}$, course *c* will not be scheduled in the last period of the day.

$$\sum_{c \in C} \sum_{r \in R} x_{crp} = 0 \qquad p \in P_{d''} \tag{7}$$

Let us suppose, period p_1 and p_2 of day *d* are fixed for lunch break, then each class will be free at least one period (p_1 or p_2 will be free).

$$\sum_{r \in R} \sum_{c \in C_k} (x_{crp_1} + x_{crp_2}) \leq 1 \quad k \in K,\ d \in D,\ p_1, p_2 \in d \tag{8}$$

For every $p \in P$, maximum number of scheduled lessons should be less than or equal to positive integer $\max_{lp}$.

$$\sum_{r \in R} \sum_{c \in C} x_{crp} \leq \max_{lp} \qquad p \in P \tag{9}$$

There are some periods in which no lesson should be scheduled. So these free periods should remain spare.

$$\sum_{r \in R} \sum_{c \in C} x_{crp} = 0 \qquad p \in P_f \tag{10}$$

3.2.3 Room related constraints

This inequality shows relationship between decision variable *z* and *x*.

$$x_{crp} \le z_{rc} \qquad c \in C, r \in R, p \in P \tag{11}$$

If room r is used for course c, it may be used for more than one period.

$$\sum_{p \in P} x_{crp} \ge z_{rc} \qquad c \in C,\ r \in R \tag{12}$$

For each room $r \in R$ and each day $d \in D$, the timetable should be compact, if room *r* is used in period p_i and in period p_j then it should be used for every period between p_i and p_j.

$$\sum_{c \in C} (x_{crp_i} - x_{crp_k} + x_{crp_j}) \le 1 \qquad r \in R\ , d \in D,\ p_i < p_k < p_j \in P_d \tag{13}$$

There are some rooms which administration wants to spare. So these free rooms should not be scheduled.

$$\sum_{p \in P} \sum_{c \in C} x_{crp} = 0 \qquad r \in R_f \tag{14}$$

Number of students taking course *c* in a room *r* at period *p* should be less than or equal to the capacity of room *r*.

$$\sum_{c \in C} s_c x_{crp} \le a_r \qquad p \in P, r \in R \tag{15}$$

3.2.4 Class related constraints

For every $k \in K$, class should not have more than two consecutive courses on each day, it means that maximum two lessons can be scheduled in a row.

$$\sum_{c \in C_k} \sum_{r \in R} (x_{crp_i} + x_{crp_{i+1}} + x_{crp_{i+2}}) \le 2 \qquad k \in K, d \in D, \quad p_i < p_{i+1} < p_{i+2} \in P_d \tag{16}$$

In class timetables, any class should not have a day with a single course (means that only one lesson is scheduled on whole day), in equation (17), w_{kd} is number of courses of class *k* in day *d* and l_{kd} counts class as having single course violations.

$$\sum_{p \in P_d} \sum_{c \in C_k} \sum_{r \in R} x_{crp} = w_{kd} \quad k \in K,\ d \in D$$

$$l_{kd} = 1 \quad \text{if} \quad w_{kd} = 1 \quad \text{and} \quad l_{kd} = 0 \quad \text{if} \quad w_{kd} \ne 1 \tag{17}$$

$$\sum_{k\in K}\sum_{d\in D} l_{kd} \qquad k \in K,\ d \in D$$

Now by summing up all classes on all days total violations can be found.

For every $k \in K$, two courses of a class must be scheduled in different periods. This is same constraint like (2) but written in a different way.

$$\sum_{c\in C_k}\sum_{r\in R}(x_{crp_i} + x_{crp_{i+1}}) \le 1 \qquad k \in K, d \in D, p_i < p_{i+1} \in P_d \tag{18}$$

No class can attend more than $l_{\max}$ teaching hours a day.

$$\sum_{c\in C_k}\sum_{r\in R}\sum_{p\in P_d} x_{crp} \le l_{\max} \qquad k \in K,\ d \in D \tag{19}$$

Here $l_{\max}$ is fixed for every class but we can use it as that each class has its own fix teaching hours and it can be written in such a way.

$$\sum_{c\in C_k}\sum_{r\in R}\sum_{p\in P_d} x_{crp} \le l_{k\max} \qquad k \in K,\ d \in D \tag{20}$$

No class can attend less than $l_{k\min}$ teaching hours a day.

$$\sum_{c\in C_k}\sum_{r\in R}\sum_{p\in P_d} x_{crp} \ge l_{k\min} \qquad k \in K,\ d \in D \tag{21}$$

All the courses of a class *k* in the day *d* must be scheduled either in the morning or in the afternoon session. For example, if class *k* attends course c_i in the morning session of day *d* $x_{c_i rp} = 1$ for some *p* such that $t_d \le p_i < l_d$ class *k* cannot attend other courses in the afternoon session $x_{c_j rp=0}$ for all $c_j \in C_k$ and $l_d \le p < t_{d+1}$ of day *d*.

$$\sum_{r\in R} x_{c_i rp_i} + \sum_{r\in R} x_{c_j rp_j} \le 1 \qquad k \in K,\ c_i, c_j \in C_k,\ c_i \ne c_j,\ d \in D \tag{22}$$

$$t_d \le p_i < l_d \le p_j < t_{d+1}$$

For each class $k \in K$, the timetable should be compact, empty periods between any two courses are not allowed.

$$\sum_{c\in C_k}\sum_{r\in R}(x_{crp_i} - x_{crp_j} + x_{crp_k}) \le 1 \qquad k \in K,\ d \in D,\ p_i < p_j < p_k \in P_d \tag{23}$$

For each class $k \in K$, class must not be scheduled on periods where it is unavailable.

$$\sum_{c\in C_k}\sum_{r\in R}\sum_{p\in P_k} x_{crp} = 0 \qquad k \in K \tag{24}$$

For each class $k \in K$, any two of its lessons should be scheduled after a gap of *g* periods.

$$\sum_{c\in C_k}\sum_{r\in R}\sum_{p\in\ q<q+1<\ldots<q+g} x_{crp} \le 1 \tag{25}$$

It occurs most of the time that some courses are taking place in one department and some others are taking place in any other department and distance between the buildings is long, so students or teachers could not reach on time after attending a course in one building to other building. So such type of courses should not be scheduled in consecutive periods. Let us suppose, c_i and c_j are two courses of the same class which take place in different long distance buildings, so these could not be scheduled on consecutive periods.

$$\sum_{p_i \in P_d} \sum_{r \in R} x_{c_i r p_i} + \sum_{p_i \in P_d} \sum_{r \in R} x_{c_j r p_{i+1}} \leq 1 \qquad d \in D \tag{26}$$

We can handle this issue by defining room types in such a way that all rooms of one building have different types from the room of other building, so these courses would be scheduled according to their types. In this way every course would be scheduled in its required building. But if the courses of different buildings occur on consecutive periods, long distance between these buildings would create a problem for students to reach on time to attend the lesson.

There are courses in which, many students have enrolled. It is very difficult to handle such a big lesson so we would prefer to divide this course in sub courses and consider each sub course as a complete course. One could also divide these big courses in many sections. Then he can accordingly write mathematical formulation. We would like to explain, how one would define sets and subsets to write such type of formulation. This division would be more beneficial if one want to put some conditions on the sections of a course particularly. The same would be the case for big classes and these big classes could also be divided in sections. Similarly one can divide big courses in sections and can accordingly write new sets to formulate problem.

A set of classes $K = \{1, \ldots, \bar{k}\}$

$s_k = \{1, \ldots, \bar{s_k}\} \qquad k \in K$

C_{s_k} = {Set of courses in sections of class k}

One can write mathematical model after defining these subsets. For example, sections of same class could not be scheduled at the same period (same teacher teaches).

$$\sum_{c \in C_{S_k}} \sum_{r \in R} x_{crp} \leq 1 \quad p \in P \tag{27}$$

3.2.5 Course related constraints

For every course $c \in C$, if course c is scheduled in day d, i.e., $u_{cd} = 1$ the number of daily hours of course c should be between $n_{c\min}$ and $n_{c\max}$.

$$\sum_{p \in P_d} \sum_{r \in R} x_{crp} \geq n_{c\min}\, u_{cd} \qquad c \in C\ ,\ d \in D \tag{28}$$

$$\sum_{p \in P_d} \sum_{r \in R} x_{crp} \leq n_{c\max}\, u_{cd} \qquad c \in C\ ,\ d \in D \tag{29}$$

For every course $c \in C$, the timetable should be compact. If more than one hours of the same course c are scheduled in day d, they have to be assigned to adjacent periods.

$$\sum_{r \in R} (x_{crp_i} - x_{crp_j} + x_{crp_k}) \leq 1 \qquad c \in C\ ,\ d \in D\ ,\ p_i < p_j < p_k \in P_d \tag{30}$$

All the hours of a course $c \in C$ scheduled in a day $d \in D$ should be located in the same room $r \in R$.

$$x_{cr_i p_i} + x_{cr_j p_j} \leq 1 \qquad c \in C\ ,\ 1 \leq r_i < r_j \leq \bar{r}\ ,\ d \in D\ ,\ p_i < p_j \in P_d \tag{31}$$

All the hours of a course $c \in C$ should be scheduled in the same room $r \in R$ for whole timetable.

$$x_{cr_i p_i} + x_{cr_j p_j} \leq 1 \qquad c \in C\ ,\ 1 \leq r_i < r_j \leq \bar{r}\ ,\ p_i < p_j \in P \tag{32}$$

For every course $c \in C$, course will not be scheduled in period p at room r, if room r will not be available at period p.

$$x_{crp} = 0 \qquad c \in C\ ,\ r \in R\ ,\ p \in P \setminus P_r \tag{33}$$

For every course $c \in C_t$ (taught by teacher t), course will not be scheduled in period p, if teacher t will not be available at period p.

$$x_{crp} = 0 \qquad t \in T\ , c \in C_t\ ,\ r \in R\ ,\ p \in P \setminus P_t \tag{34}$$

For every course $c \in A$, courses c will be scheduled in its pre assigned periods.

$$\sum_{p \in P_c} \sum_{r \in R} x_{crp} = |P_c| \qquad c \in A \tag{35}$$

For every course $c \in F$, that course c will not be scheduled on its forbidden period.

$$\sum_{p \in P_c} \sum_{r \in R} x_{crp} = 0 \qquad c \in F \tag{36}$$

For every course $c \in C$, Constraints (37), (38) and (39) guarantee minimum and maximum working days for course c constraints are satisfied.

$$\sum_{r \in R} x_{crp} \leq u_{cd} \qquad c \in C,\ d \in D,\ \ p \in P_d \tag{37}$$

$$\sum_{d \in D} u_{cd} \geq \min_{wc} \qquad c \in C \tag{38}$$

$$\sum_{d \in D} u_{cd} \leq \max_{wc} \qquad c \in C \tag{39}$$

Inequality (40) shows the relationship between decision variables *u* and *x*.

$$\sum_{r \in R} \sum_{t_d \leq p < t_{d+1}} x_{crp} \geq u_{dc} \qquad c \in C,\ d \in D \tag{40}$$

Some courses should be scheduled on the same day. Let $c_1, c_2, ..., c_j$ are courses which should be assigned on the same day.

$$u_{c_1 d} = u_{c_2 d} = ... = u_{c_j d} \qquad d \in D \tag{41}$$

Some courses should be scheduled on different days. Let $c_1, c_2, ..., c_j$ are courses which should be assigned on different days.

$$u_{c_1 d} + u_{c_2 d} + ... + u_{c_j d} \leq 1 \qquad d \in D \tag{42}$$

We can define precedence constraints among courses as in relation (43). Let $c_i < c_j$ be two courses and c_i should be scheduled before c_j for every day.

$$\sum_{r \in R} x_{c_j r p_1} + \sum_{r \in R} x_{c_i r p_2} \leq 1 \quad d \in D,\ p_1 \leq p_2 \in P_d \tag{43}$$

Sometimes, there is requirement that some courses should not be scheduled on the same period. This constraint is like class clash constraint where lessons of same class can not be scheduled in same period. Let us suppose $c_1, c_2 ... c_j$ are courses which should be scheduled on different periods.

$$\sum_{r \in R} (x_{c_1 r p} + x_{c_2 r p} + ... + x_{c_j r p}) \leq 1 \qquad p \in P \tag{44}$$

There are courses which should be scheduled on the same period. Let us suppose $c_1, c_2 ... c_j$ are courses which should be scheduled on the same periods.

$$\sum_{r \in R} x_{c_1 r p} = \sum_{r \in R} x_{c_2 r p} = ... = \sum_{r \in R} x_{c_j r p} \qquad p \in P \tag{45}$$

No course $c \in C$ can exceed more than $\max l_{cd}$ teaching hours a day.

$$\sum_{r \in R} \sum_{p \in P_d} x_{crp} \leq \max l_{cd} \qquad c \in C, \qquad d \in D \tag{46}$$

No course $c \in C$ can have less than $\min l_{cd}$ teaching hours a day.

$$\sum_{r\in R}\sum_{p\in P_d} x_{crp} \geq \min l_{cd} \qquad c \in C,\ d \in D \tag{47}$$

3.2.6 Teacher related constraints

For every $t \in T$, constraint (48) and (49) limit the number of working days for each teacher *t*. Constraint (48) speaks about role of Ψ_{td} in this relation and (49) ensures the maximum working days limit for every teacher *t*. Because some teachers are involved in administrative or non academic activities, they need some free days to do this work. Similarly one can write relation for minimum working days.

$$\sum_{r\in R} x_{crp} \leq \Psi_{td} \qquad c \in C_t,\ t \in T,\ d \in D,\ t_d \leq p < t_{d+1} \tag{48}$$

$$\sum_{d\in D} \Psi_{td} \leq e_t \qquad t \in T \tag{49}$$

The constraint (49) could be written in this way as well. Teachers should have q days free when total days in timetable are $\bar{d}$.

$$\sum_{d\in D} \Psi_{td} \leq \bar{d} - q \qquad t \in T \tag{50}$$

We can define minimum and maximum load per day for every teacher *t*. Let us suppose $\min_{lt}$ and $\max_{lt}$ are non negative integers defining minimum and maximum load.

$$\sum_{p\in P_d}\sum_{r\in R}\sum_{c\in C_t} x_{crp} \leq \max_{lt} \quad t \in T,\ d \in D \tag{51}$$

$$\sum_{p\in P_d}\sum_{r\in R}\sum_{c\in C_t} x_{crp} \geq \min_{lt} \quad t \in T,\ d \in D \tag{52}$$

For each teacher $t \in T$, any two of its courses should be scheduled after a gap of *g* periods for every day $d \in D$.

$$\sum_{c\in C_t}\sum_{r\in R}\sum_{p\in\ q<q+1<\ldots<q+g} x_{crp} \leq 1 \qquad q < q+1 < \ldots < q+g \in P_d, d \in D \tag{53}$$

Every teacher has a preference to teach in some specific rooms.

$$\sum_{c\in C_t}\sum_{r\in R-R_t}\sum_{p\in P} x_{crp} = 0 \qquad t \in T \tag{54}$$

$x_{crp} \in \{0,1\}$ $\quad p \in P,\ r \in R,\ c \in C,$

$u_{cd} \in \{0,1\}$ $\quad c \in C,\ d \in D$

$\Psi_{td} \in \{0, 1\}$ $\quad t \in T,\ d \in D$

3.3 Discussion on Objective functions

In timetabling problem we have two types of constraints, which are called hard constraints and soft constraints. Hard constraints have a higher priority than soft, and their satisfaction is usually mandatory. Solution should satisfy maximum number of soft constraints too. The usage of constraints as hard or soft depends on the demands and requirements of the institution. In one problem, a constraint used as a hard constraint it can be used as a soft constraint in other problem. Normally soft constraints are written in the objective function and penalty or desirability of any event is set by giving value to the weights.

Many objective functions can be written according to the institution requirements by using soft constraints. Here we have taken some constraints as soft constraints and have shown how objective function from these constraints can be written. We use minimum working days for course *c*, room capacity, each course should use minimum rooms i.e. if it is scheduled in some room next time it should be scheduled again in the same room, compactness for course *c* i.e., it is required that the periods assigned to day *d* for course c should be adjacent, the timetable should be compact for every class i.e. empty periods between any two courses of same class are not allowed, classes should not have a course in the last period of the day, any class should not have more than two consecutive courses on each day and any class should not have single course on any day. Now multi objective function for these soft constraints can be written in this manner.

Min z = f × F(c) + h × F(r,p,c) + i × G(c) + w × F(c,d) + l × F(k,d) + j × F(p) + k × G(k,d) + m × S(k,d)

F(c) = $\sum_{c \in C} q_c$ = Total violations for minimum working days constraint for all courses

F(r,p,c) = $\sum_{r \in R} \sum_{p \in P} \sum_{c \in C, s_c > a_r} x_{crp}(s_c - a_r)$ = Total sum of scheduled students above than room capacity

G(c) = $\sum_{c \in C} g_c$ = Total sum of minimum rooms usage violations for all courses

F(c,d) = $\sum_{c \in C} \sum_{d \in D} m_{cd}$ = Total course compactness violations done by all courses over all days

F(k,d) = $\sum_{d \in D} \sum_{k \in K} y'_{kd}$ = Total class compactness violations done by all classes over all days

F(p) = $\sum_{p \in P_{d''}} o_p$ = Total violations of occurance of classes in the last period of the day

G(k,d) = $\sum_{k \in K} \sum_{d \in D} b_{kd}$ = Total violations by all classes over all days for more than two consecutive courses constraint

S(k,d) = $\sum_{k \in K} \sum_{d \in D} l_{kd}$ = Total violations of scheduling single course of any class on any day

Where $q_c = \min_{wc} - \sum_{d \in D} u_{cd}$ $\quad c \in C$

$g_c = (\sum_{r \in R} z_{rc}) - 1$ $\quad c \in C$

$m_{cd} = \sum_{r \in R} (x_{crp_1} - x_{crp_2} + x_{crp_3}) - 1$ $\quad c \in C,\ d \in D,\ t_d \le p_1 < p_2 < p_3 < t_{d+1}$

$y'_{kd} = \sum_{c \in C_k} \sum_{r \in R} (x_{crp_1} - x_{crp_2} + x_{crp_3}) - 1$ $\quad k \in K,\ d \in D,\ t_d \le p_1 < p_2 < p_3 < t_{d+1}$

$o_p = \sum_{c \in C} \sum_{r \in R} x_{crp}$ $\quad p \in P_{d''}$

$b_{kd} = \sum_{c \in C_k} \sum_{r \in R} (x_{crp_1} + x_{crp_2} + x_{crp_3}) - 2$ $\quad k \in K,\ d \in D,\ t_d \le p_1 < p_2 < p_3 < t_{d+1}$

$\sum_{p \in P_d} \sum_{c \in C_k} \sum_{r \in R} x_{crp} = w_{kd}$ $\quad k \in K,\ d \in D$

l_{kd} =1, if w_{kd} =1 and 0 otherwise

$q_c \ge 0$, $g_c \ge 0$, $m_{cd} \ge 0$, $y'_{kd} \ge 0$, $o_p \ge 0$, $b_{kd} \ge 0$ $\quad c \in C,\ d \in D,\ k \in K,\ k \in K, p \in P_{d''}$

and

f >0 , h >0 , i>0 , w>0, l>0, j>0, k>0, m>0

The failure to satisfy constraint type is measured by the non-negative variable $q_c \ge 0$, $g_c \ge 0$, $m_{cd} \ge 0$, $y'_{kd} \ge 0$, $o_p \ge 0$, $b_{kd} \ge 0$ and is penalized via fixed parameters f, i, w, l, j, k each respectively and h is penalized fixed parameter for capacity constraint. Which constraint is more agreeable than other, these penalty parameters decide. The objective function sets to minimize the infeasibility of the soft constraints required by the scheduler.

3.4 An example of timetabling problem

Now we present an example which considers some constraints of university course timetabling problem namely all lessons of a course should be scheduled, courses of a curriculum must not be scheduled in a same period, teacher should not be double booked and room should not be double booked. With these constraints, we have two more hard constraints, course should be scheduled in the required room type and a course cannot occur in a forbidden period.

The objective function is maximizing the desirability d_{cp} of scheduling all courses. Each course has a desirability index for each period. This desirability index ranges from 0 to 4, where 4 represents most desirable period and 0 represents the least desirable period for that course. The purpose of solving this simple example is to show that how sets and subsets can be used for the timetabling problem. We think that the solution of this simple example can help to understand the procedure of the solution. The data of the example is given as follows:

Data:

d_{cp}	c=1	c=2	c=3	c=4
p=1	0	2	0	4
p=2	0	1	3	2
p=3	1	0	2	2
p=4	1	0	0	3
p=5	2	1	1	2
p=6	2	0	4	3
p=7	0	1	3	4

Table for d_{cp} C=1, 2, 3, 4 P= 1,…, 7

m_{xp}	x=1	x=2
p=1	1	2
p=2	2	1
p=3	1	2
p=4	2	1
p=5	1	2
p=6	2	1
p=7	1	2

Table for m_{xp} x=1, 2 P= 1,…,7

Set of course $C= \{1,\ldots, \bar{c}\}$, $\bar{c}$ =4,

Set of rooms $R= \{1,\ldots, \bar{r}\}$, $\bar{r}$ =4

Set of periods $P = \{1,\ldots, \bar{p}\}$, $\bar{p} = 7$

Set of classes $K = \{1,\ldots, \bar{k}\}$, $\bar{k} = 2$

Set of teachers $T = \{1,\ldots, \bar{t}\}$, $\bar{t} = 3$

Set of types of rooms $X = \{1,\ldots, \bar{x}\}$, $\bar{x} = 2$

n_c =Number of teaching hours to be scheduled for every course c,

$n_1 = 3, n_2 = 4, n_3 = 3, n_4 = 4$

C_k =Set of the courses that the class *k* should attend, $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$

C_t =Set of courses taught by teacher *t*, $C_1 = \{1\}$, $C_2 = \{2\}$, $C_3 = \{3, 4\}$

R_x = Set of rooms of type *x*, $R_1 = \{1, 2\}, R_2 = \{3, 4\}$

C_x = Set of courses requiring rooms of type *x*, $C_1 = \{1, 3\}$, $C_2 = \{2, 4\}$

P_c = Set of forbidden periods for course *c*, $P_1 = \{1,2,7\}, P_2 = \{3,4,6\}, P_3 = \{1,4\}$

F = set of courses for which there exists forbidden periods, $F = \{1, 2, 3\}$

m_{xp} = Number of rooms of type *x* available at period *p*, for every $p \in P$

d_{cp} = Desirability of scheduling the course *c* at period *p*, usually d_{cp} measures the desirability of the period *p* for the teacher *t* ($c \in C_t$).

Rooms4	2		×	4	×	×	2
3	4	2	4		2		4
2		×		×	3	1	
1	×	3	1	1		3	×
Periods	1	2	3	4	5	6	7

Table 3.1: Solution of timetabling example by CPLEX

Mathematical model of the problem can be written in this way.

x_{crp} = 1 if course $c \in C$ is scheduled in room $r \in R$ at period $p \in P$, x_{crp} = 0 otherwise;

max $\sum_{c \in C} \sum_{p \in P} d_{cp} \sum_{r \in R} x_{crp}$

$$\sum_{r \in R} \sum_{p \in P} x_{crp} = n_c \qquad c \in C \qquad \text{(a)}$$

$$\sum_{c \in C_k} \sum_{r \in R} x_{crp} \leq 1 \qquad k \in K \ , \ p \in P \qquad \text{(b)}$$

$$\sum_{c \in C_t} \sum_{r \in R} x_{crp} \leq 1 \qquad p \in P \, , \ t \in T \qquad \text{(c)}$$

$$\sum_{c \in C} x_{crp} \leq 1 \qquad p \in P \, , \ r \in R \qquad \text{(d)}$$

$$\sum_{r \in R_x} \sum_{c \in C_x} x_{crp} \leq m_{xp} \qquad p \in P, x \in X \qquad \text{(e)}$$

$$\sum_{r \in R_x} \sum_{c \in C - C_x} x_{crp} = 0 \qquad p \in P, x \in X \qquad \text{(f)}$$

$$\sum_{p \in P_c} \sum_{r \in R} x_{crp} = 0 \qquad c \in F \qquad \text{(g)}$$

We have solved this example by CPLEX. The result is shown in Table 3.1 where green block represents that the room is available but red block represents non availability of the room.

3.5 Conclusion

The university course timetabling problem is a hard problem which must be solved by departments in the beginning of the semester. It is a difficult task for which universities devote a large amount of human and material resources every year. These problems involve lot of constraints, which should be satisfied. A huge search space has to be explored, even if the size of the problem is not large enough. There is not a specific definition of this problem because each institution has its own priorities and choices. In this chapter we have given a generalized mathematical formulation for university course timetabling problem. Many constraints from different university environments have been discussed and have been written in their mathematical relations. We have given details in Chapter 5 of those different problems which have been become sub problem of our generalized model.

In the beginning of the chapter we have defined sets, subsets, different parameters and decision variables needed for formulating this generalized problem. We have classified these constraints in six sets namely, hard constraints, period related constraints, room related constraints, class related constraints, course related constraints and teacher related constraints. Hard constraints means here is that these constraints normally have been used as hard in university course timetabling problem. We have explained how the problem of big courses (means many students have enrolled) can be tackled. Similarly we have demonstrated big classes can cause problem for administration and how these can be divided in sections.

In university timetabling problem cost function has many objectives to achieve and normally all these objectives are expressed in form of unique objective function. In this chapter, we have discussed different objective functions when treating university course timetabling problem as an optimization problem and have explained how soft constraints can be used as a part of objective function. At the end, chapter is finished with an example of university timetabling problem solved by CPLEX.

Chapter 4

4 GENERIC MODEL FOR EXAMINATION TIMETABLING PROBLEM

The primary purpose of examination timetabling problems is to assign a session to a room for every examination which satisfies a given set of constraints. Each institution has its own set of constraints according to its policies and it varies from institution to institution like course timetabling problem. Each institution wants to have a good quality of examination timetable and quality of timetable would also differ from institution to institution. For some institution, a feasible timetable may be acceptable timetable but for others it does not satisfy the university required criteria.

Usually the quality of a timetable is measured by the satisfaction of soft constraints. For example a student cannot have more than one examination per day. So how many students would have examinations more than one on any day is counted as a violation and it decreases the quality of the timetable. So for an acceptable timetable there would be several different quality measures simultaneously. And the objective function is the combination of these measures with relative weights that shows their importance in the timetable. For example in some institutions, there is sufficient number of available rooms or these rooms have large capacities. So these institutions give less importance to room related constraints while scheduling.

A common constraint for educational institutions is an event clash constraint. It also occurs in examination timetabling. But due to difficult nature of examination timetabling problems, even some institutions relax this constraint. So these institutions allow a student to take two examinations at the same time and they try to resolve this conflict by quarantining. But normally it would not happen and it is very rare.

Our remaining chapter is arranged in this way. In Section 4.1, we have presented few mostly used constraints for examination timetabling problems. In Section 4.2 we have defined sets,

sub sets, parameters and decision variables for model. Section 4.3 covers linear integer programming model of the generic model. Chapter is concluded in Section 4.4.

4.1 Most frequently used constraints for examination timetabling

These constraints have been used many times by different authors.

(1) Clashing: Any student cannot have two examinations in the same periods.

(2) Total capacity: The total number of students taking exams in the same period should be less than the total number of students allowed for that period.

(3) Room capacity: Total number of students taking examinations in the same room in the same period should be less than the capacity of the room.

(4) Examination capacity: The total number of examinations scheduled in a period should be less than a defined specific number.

(5) Room availability: All rooms are not available all the time. They can be only available in some specific periods.

(6) Exam availability: Some examinations are already assigned to specific periods or can only be held in a limited set of periods.

(7) Room-Exam compatibility: Some examinations are required to be held in some specified type of rooms.

(8) Exam precedence constraints: There could be precedence constraints between examinations i.e. that some examinations should be held before others.

(9) Examination compactness: There should not be compactness in some examinations. This means that some examinations should not be scheduled in adjacent periods.

(10) Large exams: Large examinations should be scheduled earlier in the timetable (e.g. examinations with more than 500 students must be held in the first 10 sessions).

We have written some mostly used constraints as an example but in real world problems institutions use many more constraints. Some institutions use them as soft constraints and some others consider them as hard constraints [154].

4.2 Features of the generic model

In this section, we shall propose a generic mathematical formulation for examination timetabling problem.

4.2.1 Entities for the model

We have used the following entities for our generalized examination timetabling problem.

Examinations and Invigilators

Each examination has a fixed number of hours to be scheduled and invigilators have been assigned previously to each examination.

Days, Periods per day and Total Periods

We have fixed number of examination days. Each day has a specified number of periods. Total number of periods is the sum of all days periods.

Slots

A triplet of room-period-day is called one slot. Total number of slots can be found by adding all days slots.

Rooms

Each room has its own capacity (number of available seats). Each room also has a type called room type. Two rooms of same capacity can have different types.

Class

Class is a group of students attending exactly the same examinations.

4.2.2 Notations, sets, sub sets and parameters

This section is devoted for notations, sets, sub sets and parameters required to formulate mathematical model.

- A set of examination $E = \{1,\ldots, \bar{e}\}$. Each examination has a fixed number of hours, which can be scheduled during the timetable.

- A set of rooms R= {1,…, $\bar{r}$ }. Rooms can be of different types, including computer rooms and laboratories for specific examination. Each room has a fixed capacity.
- A set of periods P = {1,…, $\bar{p}$ }. P is a set of periods, which have different durations.
- A set of days D = {1,…, $\bar{d}$ }. D is the examination day of the timetable.
- A set of classes K = {1,…, $\bar{k}$ }. K is a set of classes (classes are groups of students attending exactly the same examinations).
- A set of invigilators I = {1,…, $\bar{i}$ }. Invigilators have been assigned previously to each examination.
- A set of types of rooms X= {1,…, $\bar{x}$ }.
- A set of duration types of examinations $U_e d_e$, D'= {1,…, $\bar{d}'$ }.

d_e = Duration of examination $e \in E$

d_p = Duration of period $p \in P$

t_d = First period of the morning session in day d

l_d = First period of the afternoon session in day d

$E_k \subset E$ = set of the examinations that the class k should take

$E_{so} \subset E$ = set of the examinations which are the sole occupier (means that these examinations will not share room and period with any other examination).

$\bar{P}_e \subset P$ = set of forbidden periods for examination e, for every $e \in E$

$F \subset E$ = set of examinations for which some periods are forbidden

$R_f \subset R$ = set of rooms which should be free in timetable, it means that any examination $e \in E$ should not be scheduled in any period $p \in P$, in these rooms.

$P_f \subset P$ = set of periods which should be free in timetable, it means that any examination $e \in E$ should not be scheduled in any room $r \in R$, in these periods.

$E_x \subset E$ =set of examinations requiring rooms of type x, for every $x \in X$

$R_x \subset R$ = set of rooms of type x, for every $x \in X$

C_{xp} = capacity of rooms of type x available in period p

$l_{k\max}$ = Maximum daily number of examinations allowed for every class $k \in K$

$P_d \subset P$ = set of periods for day d, *for* every $d \in D$

$E_i \subset E$ = subset of examinations supervised by invigilator i for every invigilator i

e_i = Maximum number of working days allowed for the invigilator *i*

$E_{i_i i_j}$ = Set of examinations supervised by invigilator i_i and i_j together, for every invigilator i_i and $i_j \in I$

$\max_{li}$ = Maximum load per day for every invigilator *i*

$\min_{li}$ = Minimum load per day for every invigilator *i*

$P_r \subseteq P$ = set of periods for which room *r* is available, for every $r \in R$

$P_i \subseteq P$ = set of periods for which invigilator *i* is available, for every $i \in I$

$A \subset E$ = set of pre assigned examinations (pre assigned period)

$B \subset E$ = set of pre assigned examinations (pre assigned room)

p_e = Pre assigned period for examination *e*, for every $e \in A$

r_e = Pre assigned room for examination *e*, for every $e \in B$

$P_{d''} \subset P$ = set of last periods of days

$\max_{ep}$ = Maximum examinations scheduled per period *p*

s_e = Number of students in examination *e*

a_r = capacity of room *r*

a_p = capacity of period *p*

4.2.3 Decision variables

We have defined four binary variables to formulate our mathematical model.

- x_{erp} = 1 if examination $e \in E$ is scheduled in room $r \in R$ at period $p \in P$, x_{erp} = 0 otherwise;

- $y_{d'rp}$ = 1 if duration type d' is used in room $r \in R$ at period $p \in P$, $y_{d'rp}$ = 0 otherwise;

- Ψ_{id} = 1 if $d \in D$ is a surveillance day for invigilator $i \in I$, 0 otherwise;

- u_{ed} = 1 if examination $e \in E$ is assigned to the day $d \in D$, 0 otherwise;

4.2.4 Objective function

We have presented a generalized model for examination timetabling which consists of many constraints. Many examination timetabling problems are sub problem of this generic model

(Chapter 5, Section 5.2). So we have not written any specific objective function here for this model. We have formulated mathematical relations for different constraints of timetabling problem. One can write mathematical objective function equation using this model after choosing constraints as hard and soft. We have explained the procedure of writing any objective function from these constraints in (Chapter 3, Section3.3).

4.3 Integer programming model for generalized examination timetabling problem

In this section, we present our linear integer programming model for examination timetabling problem.

For every examination $e \in E$, it should be scheduled exactly once, in unique room and in unique period.

$$\sum_{r \in R} \sum_{p \in P} x_{erp} = 1 \qquad e \in E \tag{1}$$

Sum of number of students taking examinations in a room *r* at period *p* should be less than or equal to the capacity of room *r*.

$$\sum_{e \in E} s_e x_{erp} \leq a_r \qquad p \in P, \quad r \in R \tag{2}$$

Duration of examination e scheduled in period *p* should be less than or equal to the duration of period *p*.

$$\sum_{r \in R} d_e x_{erp} \leq d_p \qquad p \in P, \quad e \in E \tag{3}$$

For every class $k \in K$, class *k* cannot take more than one examination at period $p \in P$.

$$\sum_{e \in E_k} \sum_{r \in R} x_{erp} \leq 1 \qquad k \in K \ , \ p \in P \tag{4}$$

We can define precedence constraint among examinations in this way. Let $e_i < e_j$ are two examinations and e_i should be scheduled before e_j.

$$\sum_{r \in R} x_{e_j r p_1} + \sum_{r \in R} x_{e_i r p_2} \leq 1 \qquad p_1 \leq p_2 \in P \tag{5}$$

Let us suppose $e_1, e_2 \ldots e_j$ are examinations which should be scheduled on different periods.

$$\sum_{r \in R} (x_{e_1 rp} + x_{e_2 rp} + \ldots + x_{e_j rp}) \leq 1 \qquad p \in P \tag{6}$$

There are examinations which should be scheduled on the same period. Let us suppose $e_1, e_2 \ldots e_j$ are examinations which should be scheduled on same periods.

$$\sum_{r\in R} x_{e_1rp} = \sum_{r\in R} x_{e_2rp} = \ldots = \sum_{r\in R} x_{e_jrp} \qquad p \in P \tag{7}$$

For every $e_1 \in E_{so}$, if examination e_1 is scheduled in period *p* at room *r* then e_1 would be the sole occupier.

$$x_{e_1rp} + x_{e_2rp} \le 1 \qquad e_1 \in E_{so}, e_2 \in E, e_1 \neq e_2, p \in P, r \in R \tag{8}$$

For every $k \in K$ two examinations of a class *k* should not be scheduled in a row on each day

$$\sum_{e\in E_k} \sum_{r\in R} (x_{erp_1} + x_{erp_2}) \le 1 \qquad k \in K, d \in D, p_1 < p_2 \in P_d \tag{9}$$

For each class $k \in K$, we put penalty for every two examinations, which are scheduled on same day but on non consecutive periods. One should not be confused with the notation $p_1 < p_2 < p_3 \in P_d$ this does not mean that these $p_1 < p_2 < p_3$ are consecutive integers. This means that first integer is less than second one and second one is less (not necessarily immediate less) than third one.

$$\sum_{e\in E_k} \sum_{r\in R} (x_{erp_1} - x_{erp_2} + x_{erp_3}) \le 1 \qquad k \in K,\ d \in D,\ p_1 < p_2 < p_3 \in P_d \tag{10}$$

Front load penalty constraint, which explains big examinations with respect to number of students should be scheduled earlier in the timetable because they require much time for their marking.

For every examination $e \in F$, examination *e* will not be scheduled on its forbidden period.

$$\sum_{p\in \bar{P}_e} \sum_{r\in R} x_{erp} = 0 \qquad e \in F \tag{11}$$

There are some rooms which administration wants to be spare. So these free rooms should not be scheduled.

$$\sum_{p\in P} \sum_{e\in E} x_{erp} = 0 \qquad r \in R_f \tag{12}$$

There are some periods which should not be scheduled. So these free periods should remain spare.

$$\sum_{r\in R} \sum_{e\in E} x_{erp} = 0 \qquad p \in P_f \tag{13}$$

For each class $k \in K$, any two of its examinations should be scheduled after a gap of *g* periods for every day *d*.

$$\sum_{e\in E_k} \sum_{r\in R} \sum_{p\in\ q<q+1<\ldots<q+g} x_{erp} \le 1 \qquad q < q+1 < \ldots < q+g \in P_d, d \in D \tag{14}$$

Examinations of equal length should be scheduled together.

$$\sum_{d' \in D'} y_{d'rp} \le \sum_{e \in E} x_{erp} \qquad r \in R, p \in P \tag{15}$$

$$\sum_{d' \in D'} y_{d'rp} \le 1 \qquad r \in R, p \in P$$

All the examinations of a class $k \in K$ scheduled in a day $d \in D$ should be located in the same room $r \in R$.

$$x_{e_1 r_1 p_1} + x_{e_2 r_2 p_2} \le 1 \qquad e_1, e_2 \in E_k \ , r_1 < r_2 \in R \ , \ d \in D, \ p_1 < p_2 \in P_d \tag{16}$$

No class can attend more than $l_{k\max}$ examinations a day.

$$\sum_{e \in E_k} \sum_{r \in R} \sum_{p \in P_d} x_{erp} \le l_{k\max} \qquad k \in K, \ d \in D \tag{17}$$

All the examinations of a class *k* in the day *d* must be scheduled either in the morning or in the afternoon session. For example, if class *k* takes examination e_1 in the morning session of day *d* $x_{e_1 rp} = 1$ for some *p* such that $t_d \le p < l_d$ class *k* cannot take other examinations in the afternoon session $x_{e_2 rp=0}$ for all $e_2 \in E_k$ and $l_d \le p < t_{d+1}$ of day *d*.

$$\sum_{r \in R} x_{e_1 r p_1} + \sum_{r \in R} x_{e_2 r p_2} \le 1 \qquad k \in K, \ e_1 \ , e_2 \in E_k, \ d \in D, \ t_d \le p_1 < l_d \le p_2 < t_{d+1} \tag{18}$$

Some examinations are supervised by more than one invigilator, let e_i and e_j are any two examinations which are supervised by same two invigilator i_i and i_j, if these examinations are scheduled in period p, then these will be scheduled in a same room.

$$x_{e_i r_i p} + x_{e_j r_j p} \le 1 \qquad p \in P, \ r_i, r_j \in R, \ e_i, e_j \in E_{i_i i_j}, i_i, i_j \in I \tag{19}$$

For every $i \in I$, constraints (20) and (21) limit the number of working days for each invigilator *i*. Constraint (20) speaks about role of Ψ_{id} in this relation and (21) ensures the maximum working days limit for every invigilator *i*.

$$\sum_{r \in R} x_{erp} \le \Psi_{id} \qquad e \in E_i, \ i \in I, \ d \in D \ , \ p \in P_d \tag{20}$$

$$\sum_{d \in D} \Psi_{id} \le e_i \qquad i \in I \tag{21}$$

Invigilators should have *q* days free when total days in timetable are $\bar{d}$, some invigilators are involved in administrative or non academic activities so they need some free days to do this work.

$$\sum_{d \in D} \Psi_{id} \le \bar{d} - q \qquad i \in I \tag{22}$$

We can define minimum and maximum load per day for every invigilator *i*. Let us suppose $\min_{li}$ and $\max_{li}$ are non negative integers defining minimum and maximum load.

$$\sum_{p\in P_d}\sum_{r\in R}\sum_{e\in E_i} x_{erp} \le \max_{li} \qquad i\in I,\ d\in D \tag{23}$$

$$\sum_{p\in P_d}\sum_{r\in R}\sum_{e\in E_i} x_{erp} \ge \min_{li} \qquad i\in I,\ d\in D \tag{24}$$

For every exam $e \in E$, examination will not be scheduled in period *p* at room *r*, if room *r* will not be available at period *p*.

$$x_{erp} = 0 \qquad e\in E, r\in R, p\in P\setminus P_r \tag{25}$$

For every examination $e \in E$, examination will not be scheduled in period *p*, if invigilator *i* will not be available at period *p*.

$$x_{erp} = 0 \qquad i\in I, e\in E_i, r\in R, p\in P\setminus P_i \tag{26}$$

For every $p \in P$, the capacity of examinations (number of students) scheduled in period *p* requiring room type *x* will be less than or equal to the capacity of rooms of type *x* available at period *p*.

$$\sum_{r\in R_x}\sum_{e\in E_x} s_e\, x_{erp} \le c_{xp} \qquad p\in P, x\in X \tag{27}$$

For every $p \in P$, if examination is not adequate to room type, it could not be assigned to that room.

$$\sum_{r\in R_x}\sum_{e\in E-E_x} x_{erp} = 0 \qquad p\in P, x\in X \tag{28}$$

For every examination $e \in A$, examination *e* will be scheduled in its pre assigned period.

$$\sum_{r\in R} x_{erp_e} = 1 \qquad e\in A \tag{29}$$

For every period $p \in P_{d''}$, examination *e* will not be scheduled in the last period of the day.

$$\sum_{e\in E}\sum_{r\in R} x_{erp} = 0 \qquad p\in P_{d''} \tag{30}$$

Some examinations should be scheduled on the same day. Let $e_1, e_2, ..., e_j$ are examinations which should be assigned on the same day.

$$\sum_{p\in P_d}\sum_{r\in R} x_{erp} \le u_{ed} \qquad e\in E, d\in D\ u_{e_1 d} = u_{e_2 d} = ... = u_{e_j d} \tag{31}$$

Some examinations should be scheduled on different days. Let $e_1, e_2, ..., e_j$ are examinations which should be assigned on different days.

$$u_{e_1 d} + u_{e_2 d} + \ldots + u_{e_j d} \leq 1 \qquad d \in D \tag{32}$$

For every $p \in P$, maximum number of scheduled examinations should be less than or equal to positive integer $\max_{ep}$.

$$\sum_{r \in R} \sum_{e \in E} x_{erp} \leq \max_{ep} \qquad p \in P \tag{33}$$

Sum of number of students taking examinations at period *p* should be less than or equal to the capacity allowed for that period *p*.

$$\sum_{e \in E} \sum_{r \in R} s_e x_{erp} \leq a_p \qquad p \in P \tag{34}$$

For every examination $e \in B$, examination *e* will be scheduled in its pre assigned room r_e.

$$\sum_{p \in P} x_{er_e p} = 1 \qquad e \in B \tag{35}$$

4.4 Conclusion

Examination timetabling is a well known combinatorial optimization problem. It is becoming hard to develop adequate examination timetables for educational institutions. Institutions are now introducing a wide range of courses including a number of combined degree courses. Also they are enrolling more students in many courses. Thus the institutes have to schedule the examinations which are in thousands each year. Consequently it makes examination timetabling a difficult combinatorial optimization problem and it is very complex to solve this problem by manual means. Many appropriate algorithms have been suggested in literature to solve this kind of problem. We have proposed memetic and honey bee mating algorithms to solve examination timetabling problems (Chapter 6, Sections 6.3 and 6.4).

It is difficult to give a universal definition of examination timetabling problem because the exact nature of the constraints and quality measures are unique for individual institutions. This is the motivation to discuss different examination timetabling instances used in literature (Chapter 5, Section 5.2). Thus we have proposed a generic examination timetabling model, which could be applicable across a wide range of scenarios. We have presented here a generalized model of examination timetabling problem which covers the previously described instances and many other real world constraints.

Chapter 5

5 DIFFERENT INSTANCES OF GENERIC MODELS FOR UNIVERSITY COURSE AND EXAMINATION TIMETABLING PROBLEMS

In this chapter, we have discussed about different instances from literature. The formulations of these instances can be made by our generic model for university course and examination timetabling problems. We have discussed in detail how these instances can be obtained using our model. We have also written the objective functions for these instances by using our generic model. If any instance has used different features than us then we have explained how this is different from us and how our formulation can be used to formulate it.
Before going to instances, we would discuss about mostly used hard constraints for course timetabling. We also present here some features of examination timetabling problem which makes them different from course timetabling problem.

These are the constraints ((1), (2), (3), (4): Chapter 3) which are used generally as hard constraint in course timetabling problem everywhere. Examination timetabling problems are different from course timetabling problems because each examination has a unique entity but in course timetabling each course has many lessons to schedule. In examination timetabling a teacher can supervise more than one examination at the same period and one room can host more than one examination at the same period, which is different from course timetabling problem.

5.1 Instances of our generalized course timetabling problem

These instances are the part of our generalized university course timetabling problem and their details could be found in Sections 5.1.1, 5.1.2, and 5.1.3.

1. Pasquale Avella and Andigor Vasil’ev model
2. Post enrollment based course timetabling

3. Curriculum based course timetabling problem
4. Original timetabling problem of Udine University

There are some constraints in our generalized model which are the part of any other university problem different from previous scenarios. Their details could be found in Section 5.1.4. Some constraints which are the new addition in literature could be found in Section 5.1.5.

Different instances which are the part of our generalized examination timetabling problem could be found in Section 5.2.1, 5.2.2, 5.2.3. Details of the used constraints different from previous instances and the list of newly added constraints in generalized model could be searched in Section 5.2.4, 5.2.5. The chapter is concluded in Section 5.3.

5.1.1 Pasquale Avella and Andigor Vasil'ev model

Pasquale Avella and Andigor Vasil'ev [13] article on cutting plane algorithm for course timetabling problem had used many constraints first time. They described a case study where branch and cut algorithm was used. They transformed course timetabling as a set packing problem with side constraints. They showed a relation between course timetabling problem and the set packing problem. This relation was used to get timetabling problem inequalities from the polyhedral description of the set packing polytope, whose structure was widely studied.

In fact this was an attempt to transform course timetabling problem to set packing problem. A set packing problem could be transformed to stable set problem by making intersection graph. Let A be a matrix of size $m \times n$, whose entries are 0 or 1. Let y be a set of n variables and c be a cost vector. Then set packing problem can be defined as max $c^t y, Ay \leq 1$, where value of *y* can be 0 or 1. Intersection graph G (V, E) can be made by associating a node $e_i \in V$ with each column of A. The edge $e_i e_j$ belongs to E iff columns e_i and e_j of A are not orthogonal. They also studied the polyhedral structures of the problem in according with set packing polytope, in this way two families of cutting planes clique and lifted odd hole inequalities were derived.

Now we describe the constraints they used. They used the four hard constraints, these are the hard constraints which are used as hard almost in every institution. Numbering in bracket of the forth coming paragraph is representing the constraint number of our generalized

formulation (Chapter 3) which shows that how the problem of Pasquale becomes sub problem of our generalized course timetabling problem of Chapter 3.

All hours of a course per week must be scheduled (1).

Any class cannot attend more than one course at any period (2).

Teacher *t* cannot teach more than one course at any period (3).

Any room cannot host more than one course at anyperiod (4).
These constraints were used as soft constraints by them.

Daily hours of course *c* in a day *d* should be between $n_{c\min}$ and $n_{c\max}$ (28) and (29).
For every course the timetable should be compact. If more than one hours of the same course *c* are scheduled in day *d*, they have to be assigned to adjacent periods (30).

No class can attend more than $I_{\max}$ teaching hours a day (19).

All the courses of a class in a day *d* must be scheduled either in the morning or in the afternoon session (22).

For each class the timetable should be compact, empty periods between any two courses are not allowed (23).

Teacher can not work more than maximum number of working days allowed for him (49)
Course should be scheduled in a room which is adequate for it (5) and (6).

A course will not be scheduled in a period at room, if room is not available at that period (33).

A course will not be scheduled in a period, if teacher is not available at period (34).

5.1.2 International timetabling competition 2007 (ITC 2007)

Timetabling competition was sponsored by PATAT and WATT. This timetabling competition contained three tracks for competition. One track was about examination timetabling problem and other two were about course timetabling. From these two course timetabling, first was post enrolment based course timetabling and second was curriculum based course timetabling problem. Post enrolment based course timetabling means that firstly students are enrolled in courses and then all these courses are scheduled in such a way that all students can attend all their enrollments. But in curriculum based course timetabling scheduling is done on the basis of curricula published by university and not on the basis of enrolments of students. The third track was about examination timetabling. This Competition was organised and run by the Event Management and Planning Research Group (eventMAP) at Queen's University with partners from Cardiff University, Napier University, University of Nottingham and the University of Udine.

Educational timetabling has become a part of competition and first timetabling competition (ITC-2002) was organized by International Metaheuristic Network, where 24 participants presented feasible solutions for presented datasets from all over the world. Information about definition, rules, datasets and solution evaluation are available on website: http://www.idsia.ch/Files/ttcomp2002/. A specific problem model was proposed for the competition and formulation of this model contained many characteristics found in certain Universities. Datasets to use for competition was generated artificially and now these datasets have become standard within the research area. Many researchers had used them in their scientific works [15, 16, 17, 18]. There is a positive effect of ITC-2002 for creating a ground for cross fertilization of ideas within researchers in the timetabling community.

The Second International Timetabling Competition (ITC-2007) was started on 1st August 2007 and it was on the pattern of the first edition ITC-2002. It also added some more aspects and feature in it [42].

5.1.2.1 Post enrollment based course timetabling

This was a track in timetabling competition 2007 and this was the extension of international timetabling competition 2002. This model was about the real situation where students are given choices to attend lessons according to their wish and time table was scheduled after

students had given their choices. This model was based on the model of 2002 competition which was held in conjunction of PATAT and Metaheuristic Network.

In original model, the number of events must be scheduled in limited number of rooms when satisfying two types of constraints. These were hard and soft constraints, hard constraints are those for which the fulfillment is obligatory and soft constraints are those for which the fulfillment is optional. The solution quality depends mainly on the satisfaction of soft constraints.

5.1.2.1.1 Used constraints and difference with ITC 2002

Formally this competition problem was defined in this way. These were the constraints which were used as hard constraint in the problem model.

1. No student can attend more than one event at the same time.
2. Every event must be scheduled in a room whose capacity is less than or equivalent to the capacity of room and should be of required type.
3. Only one event should be scheduled in a room at the same period.
4. Events should be scheduled on available periods because all the periods are not available all the time.
5. Precedence constraints amongst events should be satisfied.

The last two hard constraints were added additionally in International timetabling competition 2007 (ITC 2007) and all other hard constraints were same. With these five constraints, three soft constraints were added in this problem;

1. Students should not attend events in the last period of the days.
2. Students should not attend three successive events in the same day.
3. Students should not attend only single event on any day.

These three soft constraints were same, as these are used in 2002 competition.
This mathematical model can be written by using these constraints of our mathematical formulation of Chapter 3, where constraints (2), (5), (6), (15), (4), (10), (43) are expressing hard constraints and (16), (17), (7) are expressing soft constraints.

We can write objective function for this model by using constraints (16), (17), (7). Objective function can be written as follows.

For every $k \in K$, class should not have more than two consecutive courses on each day.

$$u_{kd} = \sum_{c \in C_k} \sum_{r \in R} (x_{crp_i} + x_{crp_{i+1}} + x_{crp_{i+2}}) - 2 \qquad k \in K, d \in D, \quad p_i, p_{i+1}, p_{i+2} \in P_d \tag{16}$$

where $U_{i,i+1,i+2}\, u_{kd}$ is the sum over all the periods $i, i+1, i+2$.

In class timetables, any class should not have a day with a single course (means that only one lesson is scheduled on whole day).

$$\sum_{p \in P_d} \sum_{c \in C_k} \sum_{r \in R} x_{crp} = w_{kd} \qquad k \in K,\ d \in D$$

$$l_{kd} = 1 \quad \text{if} \quad w_{kd} = 1 \quad \text{and} \quad l_{kd} = 0 \quad \text{if} \quad w_{kd} \neq 1$$

(17)

For every period $p \in P_{d''}$, course c will not be scheduled in the last period of the day.

$$\sum_{c \in C} \sum_{r \in R} x_{crp} = b_p \qquad p \in P_{d'} \tag{7}$$

Min z = a $\sum_{k \in K, d \in D} U_{i,i+1,i+2}\, u_{kd}$ + b $\sum_{d \in D} \sum_{k \in K} l_{kd}$ + c $\sum_{p \in P_{d'}} b_p$

The failure to satisfy constraint type is measured by the non-negative variable $U_{i,i+1,i+2} u_{kd} \geq 0$, $l_{kd} \geq 0$, $b_p \geq 0$ and is penalized via fixed parameters a, b, c respectively.

5.1.2.1.2 Difference of our model with Post enrolment based course timetabling

In our model, we are using classes and courses but in competition model, they used events and students taking these events because this is a post enrollment model and in this model firstly students enroll in events then there is scheduling. We can consider each event like an examination and can make a model (examination timetabling model) or simply for understanding we can say that each course has only one lesson to schedule. Like examination timetabling each event is a unique entity means that when it is scheduled, it is finished, opposite to course timetabling where each course has some lessons. We have written all requirements in our model, we are using classes but in post enrollment problem each event has specific students, one know specifically which students are attending this event but in our

model we have classes (set of courses taken by some common students) but we do not know who these students are.

We have to modify data to use this model, firstly make curriculum from data and then solve it according to this way. We can solve all the hard constraints and one soft constraint by reforming sets.

Let us suppose, we have

S= Set of students= {1, 2, 3, 4, 5, 6, 7, 9, 10}

E = Set of events

$E = \{ e_1, e_2, e_3, e_4, e_5 \}$

Set of students enrolled in each event are as follows.

$S_{e_1} = \{1,2,5,9\},\ S_{e_2} = \{7,4,10\},\ S_{e_3} = \{3,4,6\},\ S_{e_4} = \{6,7\},\ S_{e_5} = \{2,7,10\}$

Now we can make classes in this way, take event 1 and see all its students as {1,2,5,9} are taking this event, then all the events which have any student from {1,2,5,9}, is one set of events including event 1. This set is called one class. Similarly see for events 2, 3, 4 and 5, then take all these sets and delete the sets which are repeating (just take one set from repeated sets). These all sets are set of classes for that problem. For our example, classes are as follows.

$k_1 = \{ e_1, e_5 \},\ k_2 = \{ e_2, e_3, e_4, e_5 \},\ k_3 = \{ e_2, e_3, e_4 \},\ k_4 = \{ e_1, e_2, e_4, e_5 \}$

First soft constraint (students should not have events scheduled in the last period of the day) can easily be handled. If event of class is scheduled in the last period of the day, any other event of the class could not be scheduled in this period according to definition. So numbers of students in that event are the students scheduled in the last period.

If we look at second soft constraint according to our model then the situation is that no class can attend more than two of its courses consecutively, so common students can not come more than twice of a class. But when there will be violation of consecutiveness then the common students of all these classes will be counted and these students will violate soft

constraint and soft constraint could not be scheduled directly using sets and subsets of students. Problem will be solved if to take it as soft constraint.

Students should be scheduled in this way that they should not have single event on any day, this constraint satisfaction is also difficult. One class is schedule lonely on any specific day, so according our model this is violation of soft constraint but according to student based model (Post enrollment based problem), this is not violation. Because for example, if one class is scheduled lonely on a day and another class has also been scheduled lonely on the same specific day but these two different classes can have common students, so according to post enrollment based problem this is not violation.

It is quite possible in universities that some rooms are very large and they can be partitioned in to sub rooms (partitioning is possible within them). For example a room *r* is used as a big room in period p_i but is partitioned into five small rooms in the next period p_{i+1}. Thus one can decompose this into two different room types and different room capacities. So it is quite easy to put this feature in our model because rooms are not uniformly available in our model, we can say in period *p* one room of type *x* is available and in period p+1 room of type *x* is not available but five rooms of type *y* are available.

5.1.2.2 Curriculum based course timetabling problem

Curriculum based course timetabling problem is a track in ITC 2007 and this track is about weekly scheduling where university publishes curricula first. This model applied to University of Udine (Italy) and many other Italian universities. Datasets were taken from these universities and were modified little to make them general to use them as competition datasets. The problem consists of the following entities, number of days, number of periods per day, number of courses, number of teachers, number of rooms, capacity of each room, number of curricula and many other constraints. Now we would like to express thoroughly the competition problem.

Here we present one of its tracks namely the curriculum-based course timetabling. We shall talk about the features of the model and the datasets used together with the evaluation

procedure. One can see rules to obey for competition on webpage: http://www.cs.qub.ac.uk/itc2007/.

5.1.2.2.1 Evaluation procedure for solutions

Now we shall describe how the solutions are evaluated. Let m be the total number of early and late datasets and k be the number of participants who are going to submit their solutions for all m datasets. Let X_{ij} be the result of the participant *i* for dataset *j*. Each result X_{ij} is given in form of pair (d, s) where *d* is the distance to feasibility and *s* is the score of the objective function. Firstly results were compared on the basis of the value of *d* but when competitors have same *d* value then results were compared on the basis of the objective function value.

Each participant results were transformed by using transformation matrix for each value of x_{ij} . Then for dataset j supplied values will be $x_{1j}, x_{2j}...x_{kj}$. All these values were compared with each other and rank 1 was given to the smallest observed value, rank 2 was given to second smallest observed value and so on up to the rank k. They used average ranks in case of ties.
Consider the following example where there are k = 7 participants and m = 6 datasets. Let X be the matrix of Table 5.1, the computed rank matrix R is reported in Table 5.2. By using these values, mean values of all participants was calculated and on the basis of mean value 5 finalists were selected with the lowest mean ranks. If there is a tie on last position then all the last participants of equal mean value were selected. In this case finalists can be more than five. In this example mean ranks can be seen in last column of the rank Table 5.2. Thus according to this example, finalists would be solvers 1, 4, 5, 6, and 7.

Dataset	1	2	3	4	5	6
Solver 1	(0,34)	(0,35)	(1,42)	(1,32)	(0,10)	(0,12)
Solver 2	(3,32)	(1,24)	(1,44)	(1,33)	(0,13)	(0,15)
Solver 3	(0,33)	(0,36)	(2,30)	(5,12)	(1,11)	(0,17)
Solver 4	(0,36)	(0,32)	(1,46)	(1,32)	(0,12)	(0,13)
Solver 5	(0,37)	(0,30)	(1,43)	(1,29)	(0,9)	(0,4)
Solver 6	(2,68)	(0,29)	(1,41)	(0,55)	(0,10)	(0,5)
Solver 7	(0,36)	(0,30)	(2,43)	(0,58)	(0,10)	(0,4)

Table 5.1 : Score matrix

Their runs were checked with the submitted seed to make sure that they can repeat. If these were not repeatable then other participant next to him was chosen for the top five. Then the final was conducted between these finalists. Final was conducted in this way.

1. All datasets with hidden ones were used.
2. The solvers were run by organizers.
3. For each dataset, 10 independent trials were conducted by providing solvers with a sequence of random seeds (the same was done for all solvers).

They proceeded as earlier by calculating ranks and averaging them on all trials on all datasets. The final winner of the competition was the finalist with the lowest mean rank and in case of tie more additional trials were added for all datasets until a single winner was found.

Dataset	1	2	3	4	5	6	Mean rank
Solver 1	2	5	2	4.5	3	4	3.42
Solver 2	7	7	4	6	6	6	6.00
Solver 3	1	6	6	7	7	7	5.67
Solver 4	3.5	4	5	4.5	5	5	4.50
Solver 5	5	2.5	3	3	1	1.5	2.67
Solver 6	6	1	1	1	3	3	2.50
Solver 7	3.5	2.5	7	2	3	1.5	3.25

Table 5.2: Rank matrix and mean ranks

Only feasible solutions were accepted in ITC-2002 because there it was purposely easy to produce feasible solutions for all datasets. In ITC-2007 organizers relaxed the condition for feasible solution so solvers can submit their solutions with few infeasible solutions as well, although all the datasets had at least one feasible solution. In case of infeasible solution for some datasets from different solvers comparison was conducted on an evaluation based ranking of solutions on each dataset, rather than on the actual scores. An infeasible solution on one dataset did not necessarily prejudice the overall performance of the participant due to this scoring based on rankings. Datasets were collected from larger set of interesting case which was coming from real world problem without the limitation of easy feasibility.

In ITC-2002 ranking was fully based on the solution provided by participants because for each single trial maximum CPU time was granted to each solver. So solver was able to run as many trials as he could and report only the best of all of them.In ITC-2007, Finalist solvers run on organizers machine (with new seeds), hidden datasets were used to stop this approach. Moreover in the case of stochastic algorithms, this fostered the design of robust solvers. In ITC-2002, organizers also tested the few best algorithms on unseen datasets and indeed results were good as were good for known datasets.

Time was fixed to solve datasets for competition because organizers wanted to remove degree of variability from the scoring system. This was a main question, what is a good feasible fixed amount of running time for the actual timetabling. Timetabling is done few times in a year in educational institutions so one can think that more than 10 minutes would be a reasonable grant of time. In real case timetabling many researchers have pointed out that solution of real problem is an interactive process during which a lot of datasets should be solved. Constraints and objectives are manually adjusted between runs of a working session for one single case due to live situation or last minute changes, etc. Thus a long to solve datasets can be tiresome for human operator. Although, organizers decided maximum 10 minutes time to solve a dataset.

5.1.2.2.2 Used sets for the problem

There are following entities for this problem.

Days, Periods per day and Total periods

There are 5 to 6 days per week and equal number of periods for each day. Total number of periods can be found by multiplying number of days with number of periods per day.

Courses and Teachers

Each course has fixed number of lessons to schedule and it is attended by number of student. A teacher will teach this course and each course should be scheduled for specified minimum number of working days. Moreover, there are some forbidden periods for each course.

Rooms

Each room has its capacity and each course can be scheduled to each room if it satisfies the capacity constraint.

Curricula

A curriculum is set of courses which have common students, so these courses can not be scheduled in a same period. Moreover there are also some soft constraints on curricula.

5.1.2.2.3 Purpose of the competition

There are total 21 datasets, 7 datasets are early datasets, 7 datasets are late datasets and 7 datasets are hidden datasets. All datasets are coming from University of Udine. If all hard constraints are satisfied in any solution, it is called feasible solution and all these 21 datasets have at least one feasible solution but optimal values for the soft constraints were not known.
The motivation of this competition track was to make bridge between practice and research, these datasets have certain degree of complexity in this problem so that new formulations could be brought near to those of real world problems (where data is coming from real world)[5].
The second innovation aimed at bridging the gap between research and practice: the competition introduced a significant degree of complexity in all tracks so that the new formulations employed are closer (in more aspects, although not all) to those of 'real world' problems [5] and data was coming from the real world.

5.1.3 Original timetabling problem of Udine University

Some of the participants criticized on these datasets that these are not properly designed because these do not use the most important cost component or these do not penalize the right patterns. To answer these questions organizers added some more features in the formulation [19] and main purpose was to make a formulation which could be accepted by a larger community of researchers. So this article [19] was accompanied by a web site (http: //tabu.diegm.uniud. it/ctt/) from which all information about datasets or problem description could be found. They added the following new features in the Curriculum based course timetabling problem model. These constraints can be written from our formulation of Chapter 3 by using these constraints (23), (20), (21), (26), (5), (6), (30).

Windows (Curriculum Compactness)

There should not be windows (i.e., periods without teaching) among lessons belonging to a same curriculum.

Curriculum Min, Max Load

The number of daily lessons for each curriculum should be within a given range. Each lesson below the minimum or above the maximum limit counts as one violation.

Travel Distance

Sometimes students have to move from one building to another and between these two buildings there is a long distance. So such type of lessons should not be scheduled on consecutive periods.

Room Suitability

Each course should be assigned a room which has all necessary equipment (projector, computer lab, etc.) required by course. Each lesson of a course in an unsuitable room counts as one violation.

Double Lessons

Some courses are required that their lessons scheduled in the same day are adjacent to each other. So for these courses non grouped lessons are not allowed.
Hard constraints

1. Each course has a specific number of lessons; all these lessons must be scheduled to distinct periods.
2. A room can not host more than one lesson in the same period.
3. A teacher can not teach more than one lesson in the same period.
4. Lessons of courses in the same curriculum should be scheduled in the distinct periods.
5. A lesson of a given teacher should not be scheduled in a period if teacher is unavailable in this period.

Soft constraints used in this model are as follows.

1. Number of students in a course should be less than or equal to the room sitting capacity which is hosting it. Each student above capacity counts point of penalty.
2. Each course should be spread over minimum number of working days, each day below the minimum working days counts point of penalty.
3. Lessons of the same curriculum scheduled in the same day should be adjacent to each other. Each lesson not adjacent to others counts point of penalty.
4. All lessons of a course should be delivered in the same room. If more than one room is used for a given course, there will be penalty.

Mathematical model of curriculum based course timetabling problem can be written by using constraints (1),(4),(3),(2),(34),(15),(38),(23),(32) of Chapter 3.

We can write objective function by using constraints (15), (38), (23), (32), which will minimize the objective function value.

Number of students taking course *c* in a room *r* at period *p* should be less than or equal to the capacity of room *r*. Penalty can be counted by using this equation.

$$u_{pr} = \sum_{c \in C} x_{crp}\, s_c - a_r > 0 \qquad p \in P, \quad r \in R \tag{15}$$

For every course $c \in$ C, this equation will be used in objective function to count minimum working days penalty.

$$q_c = \min_{wc} - \sum_{d \in D} u_{cd} \qquad c \in C \tag{38}$$

For each class $k \in K$, the timetable should be compact, empty periods between any two courses are not allowed.

$$y'_{kd} = \sum_{c \in C_k} \sum_{r \in R} (x_{crp_i} - x_{crp_j} + x_{crp_k}) - 1 > 0 \qquad k \in K, \quad d \in D, \quad p_i < p_j < p_k \in P_d \tag{23}$$

Where $U_{ijk} y'_{kd}$ is the *sum* over all the periods *i, j, k.*

All the hours of a course $c \in C$ should be scheduled in the same room $r \in R$, for whole timetable.

$$x_{cr_i p_i} + x_{cr_j p_j} \leq 1 \qquad c \in C, 1 \leq r_i < r_j \leq \bar{r}, \quad p_i < p_j \in P \tag{32}$$

But to calculate the penalty for single room use, we use this simple relation

$$g_c = (\sum_{r \in R} z_{rc}) - 1 \qquad c \in C$$

$$\text{Min z} = \text{f} \sum_{c \in C} q_c + \text{h} \sum_{p \in P, r \in R} u_{pr} + \text{i} \sum_{c \in C} g_c + \text{l} \sum_{d \in D} \sum_{k \in K} U_{ijk} y'_{kd}$$

The failure to satisfy constraint type is measured by the non-negative variable $q_c \geq 0$, $g_c \geq 0$, $y'_{kd} \geq 0$, $u_{pr} \geq 0$ and is penalized via fixed parameters f, h, i, l respectively.

5.1.4 Constraints different from previous scenarios

These constraints are not the part of previously written scenarios. These constraints are the part of our generalized model and could be found in front mentioned articles.

1. Some courses have been pre assigned [160].
2. Courses will not be scheduled on its forbidden periods [160].
3. No course can attend more than a specific teaching hours a day [13].
4. No course can attend less than specific teaching hours a day, if course is scheduled on that day [13].
5. Define maximum load per day for every teacher [161].
6. Each class have free period for lunch on a specific time interval [24].

5.1.5 Constraints not found in the literature

These constraints are also part of our mathematical formulation but we could not find them in literature. We can not surely say that these constraints have been used in the literature or not. But as far as we know we could not search them in any research work.

1. Room's timetable should be compact for every day. If any room is used in period p_i and in period p_j then it should be used for every period between p_i and p_j.
2. In some rooms no course should be scheduled.
3. Timetable should maintain a gap between any two courses of a class on any day.

4. Maximum working days allowed for course should be respected.
5. Some course should be scheduled on the same day. For example, set of same day courses contains two courses; if lesson of first course is scheduled on any day then lesson of second course should also be scheduled on the same day.
6. Some course should be scheduled on different days.
7. Some courses should not be scheduled on the same period.
8. There are courses which should be scheduled on the same period.
9. Define minimum load per day for every teacher, if teacher is teaching on that day.
10. Timetable should maintain a gap between any two courses of a teacher on any day.
11. Some teachers can have preference to teach in specific rooms.
12. Each period can not host lessons more than maximum number of lesson's limit.

5.1.6 Contradictory constraints for course timetabling problem

We present here some constraints which contradict each other and can not be used at the same time. These constraints have been used in different educational environments according to their requirements. Now we shall discuss them in detail. The numbering in front of each constraint is same as used in Chapter 3.

The number of courses scheduled in period *p* requiring room type *x* will be less than or equal to the number of rooms of type *x* available at period *p*. (5)

If course is not adequate to room type, it could not be assigned to that room. (6)

If one uses (5) constraint as soft constraint then constraint (6) could not be used, constraint (6) only will be used when room type constraint is used as hard constraint.

Any class should not have more than two consecutive courses on each day. (16)

Two courses of a class could not be scheduled together. (18)

Any two lessons of a class should be scheduled after a gap of *g* periods. According to this constraint two courses could not be scheduled together as well. (25)

Constraint (16) expresses that class should not have more than two consecutive courses on each day; it means that two consecutive courses are allowed but constraint (18) explains that two courses of a class could not be scheduled together. These two constraint contradict each other so could not be taken in same timetabling problem.

Timetable should be compact for each class; empty periods between any two courses are not allowed. (23)

If one uses curriculum compactness (23) then the constraints (25), (18), (16) could not be used simultaneously, as these are against each other.

Some courses are taking place in one department and some others are taking place in any other department and distance between the buildings is long, so students or teachers could not reach on time after attending a course in one building to other building. So such type of courses should not be scheduled in consecutive periods.

Constraint (26) will disturb curriculum compactness so when one is using curriculum compactness and there are also some long distance buildings, then compactness constraint could be dealt by any other way.

Course compactness (27) can be used with curriculum compactness because each course is a part of any curriculum.

In (30), (31) some timetables rooms, periods, teachers are always available and some environments these are unavailable on some occasions but unavailability on some occasions is more general case.

Course will not be scheduled in period *p* at room *r*, if room *r* is not available. (33)

Course will not be scheduled in period *p*, if teacher *t* is not available. (34)

Course will not be scheduled on its forbidden period. (36)

There are some periods which should not be scheduled. So these free periods should remain spare. (10)

In case of (33), (34), (36) and (10) course would not be scheduled but reason to not schedule is different. Course (36) would not be scheduled on forbidden periods; this constraint is different from (10) which expresses some periods are unavailable. Constraint (36) means that some periods are unavailable for course only but not for all courses.

5.2 Examination Timetabling Instances

Examination timetabling problem is widely studied problem of educational timetabling problem and a variety of different benchmark instances have been used by researchers for meaningful scientific comparisons and the exchange of research achievements [128]. Here we have discussed about the used constraints and their difference from other bench mark instances. We express that these constraints have been covered in our generalized model and these formulations are the subset of our generalized mathematical formulation. Most common and well known instances in literature are as follows.

5.2.1 University of Toronto Benchmark instances

Carter et al. [82] introduced a set of 13 real world examination timetabling problems; three from Canadian highs schools, five from Canadian universities, one from American university, one from British university and one from a Saudi Arabian university. These have been widely used as standard in examination timetabling research. Hard constraints for this problem were to schedule all examinations without conflict. They defined conflict matrix C, where each element $C_{ij} = 1$ if examination i conflicts with examination j (have common students), or $C_{ij} = 0$ otherwise. The conflict density was the ratio between the number of elements of value "1" to the total number of elements in the conflict matrix. A student must not sit for more than one examination at any period was used as hard constraint. The objective function was to minimize the number of periods needed and the number of examinations of a student scheduled in the same day consecutively**.**

Burke et al. [50] modified the Toronto benchmark instances by considering the maximum room capacity per period and objective was to minimize the students sitting into two consecutive examinations on the same day. They introduced new examination timetabling datasets coming from University of Nottingham. These were used later by a number of

researchers to test different approaches and are called University of Nottingham Benchmarks. Burke et al. [153] further modified the above problems by considering also consecutive examinations overnight.

5.2.2 University of Melbourne Benchmark instances

Merlot et al [154] introduced examination timetabling datasets at the PATAT conference in 2002 coming from the University of Melbourne. The objective for these datasets was t**o** minimize the number of occasions of students having two examinations consecutively either on the same day or overnight. For some of the examinations some periods were unavailable.

5.2.3 ITC- 2007 Benchmark instance (Examination Timetabling Track)

The 2nd International Timetabling Competition (ITC2007) has three tracks; one on examination timetabling and two on course timetabling. We have discussed tracks on course timetabling in previous section (Section 5.1.2). Here we have described the examination timetabling track introduced as part of the competition [40, 42].

These two constraints are used as required constraints in these instances. Required constraints meant that if solution did not satisfy these constraints, solution was rejected outright.

1. Every examination is assigned to at most one room.
2. Every examination is allocated to at most one period.

These were the hard constraints for the given problem.

3. Every examination is assigned to at least one room and at least one period
4. The capacity of individual rooms is not exceeded at any time throughout the examination
5. Duration of examination scheduled in period should be less than or equal to the duration of period.
6. In any period, any student is taking at most one examination.
7. Precedence constraints between examinations should be satisfied.
8. Specified pair of examinations must be scheduled in the different period.
9. Specified pair of examinations must not be scheduled in the same period.

10. Some specified examinations must be the sole occupier; it means that if this type of examination is scheduled in any period, then any other examination can not be scheduled in this period.

These constraints were used as soft constraints.

11. Two distinct examinations of a same student should not be scheduled in consecutive periods on same day.
12. Two distinct examinations in non-consecutive periods of a same student should not be scheduled on same day.
13. For every student any two of its examinations should be scheduled after a gap of g periods for every day.
14. In any period, all the examinations of same duration should be scheduled.
15. There are some rooms which administration wants to be spare. So these free rooms should not be scheduled.
16. There are some periods which should not be scheduled. So these free periods should remain spare.

These used constraints can be mathematically expressed by using these 14 equations (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(12),(13),(14),(15) of our generalized examination timetabling formulation of Chapter 4. Constraints (1), (2) and (3) of this ITC-2007 track can be written by using equation (1) of our mathematical formulation.

5.2.4 Constraints different from previous benchmark instances

These constraints are different from previously written bench mark instances. We have written a reference in front of each constraint which shows that this constraint was included in mentioned article. This does not mean that this constraint was used only in that specific article, this constraint can be part of many other articles but for convenience we have mentioned only one reference. Some of these constraints were used exactly as we have used and some of them were used with little modification according to the scenario. These constraints are given as follows.

1. All the examinations of a class should be located in the same room on any day [164].
2. No class can attend more than a specific number of examinations in a day [162].

3. All the examinations of a class in a day must be scheduled either in the morning or in the afternoon session [163].
4. Invigilators are not always available [162].
5. Number of students scheduled in a period *p* requiring room type *x* will be less than or equal to the capacity of rooms of type *x* available at period *p* [154].
6. Every examination should be assigned required room type [154].
7. Pre assigned examinations should be scheduled in their pre assigned periods [154].
8. Some examinations should be scheduled on different days [163].
9. A period can not host examinations more than maximum number of examination's limit for that period [165].
10. Sum of number of students taking examinations at period *p* should be less than or equal to the capacity allowed for that period *p* [165].

5.2.5 Constraints not found in the literature

These constraints may have been used in literature but we could not find them in literature as a part of any examination timetabling problem.

1. Examinations supervised by more than one invigilator scheduled in a same period should be assigned same room.
2. Time table should follow minimum and maximum load per day for a invigilator.
3. Invigilators should be given some free days.
4. No examination should be scheduled in the last period of the day.
5. Some examinations should be scheduled on the same day.

5.3 Conclusion and Discussion

In this Chapter, we have discussed different instances of course and examination timetabling problems which are sub part of our generalized course and examination timetabling problems. We have written the mathematical models of these instances by using our generalized examination and course timetabling problem. We have discussed in detail curriculum based course timetabling problem of (ITC-2007) international timetabling competition held in 2007, because we use these competition datasets for our experimental work in forth coming (Chapter 7). We have also demonstrated post enrolment based course timetabling problem of the same competition. We have shown the main difference of this problem with our

generalized problem and how one can modify this problem to use our generalized mathematical model. We have also discussed different constraints of university timetabling problems which are contradicting with each other and hence can not be the part of a same problem instance. During our discussion we have highlighted the real world problem of Udine University from which curriculum based course timetabling problem was made. We have written the mathematical model of this problem from our generalized model as well. We have also discussed many constraints which are added by us in model. It means that we could not found them in literature.

We have also discussed different examination instances used in literature, which are part of our generalized examination timetabling problem in detail at the end of this chapter. We have shown that how mathematical models of these instances can be deduced from our generalized model. At the end we have presented some constraints which are added by us in this newly proposed generalized examination timetabling problem.

Chapter 6

6 PROPOSITION OF RESOLUTION METHODS

In this chapter we have proposed our algorithms which could produce effective solutions for generalized timetabling problems. We have proposed two population based alogorithms named memetic algorithm and honey bee mating algorithm. Memetic algorithm uses a local search to improve the quality of solutions. Neighbourhoods used by this local search are proposed in this chapter as well. We have explained the working procedure of these algorithms in detail and also have discussed the benefits of using different operators in these algorithms. We have highlighted different search space issues and have talked about the beneficial aspects of our chromosome representation.

This chapter is arranged as follows. In Section 6.1, we have presented our chromosome representation used in both of our algorithms. We have spoken about search space issues. We have also expressed our ideas to arrange data and for the use of adequate chromosome representation to enhance the search space. In Section 6.2, we have demonstrated the procedure of making initial solution for algorithms. We have discussed different heuristics and repair strategies to construct these solutions. In Section 6.3, we have discussed our memetic algorithm. We have presented different genetic operators, infection and replacement procedures. In this section, we have also explained procedure of our proposed local search and memetic algorithm. Section 6.4 is devoted for honey bee mating algorithm. Section 6.5 is reserved for termination criteria of these algorithms and chapter is concluded in Section 6.6.

6.1 Solution representation of Algorithms

The timetable is a collection of each room timetable, where a room timetable is a two dimensional array as shown in Figure 6.1. If no lesson is booked in any period, it is called null booking which has value zero. Maximum one event can be assigned to any place in the matrix. Every timetable stores information that which lesson is placed in which room at what time on which day of the week, each booking (each cell in a matrix) is one gene. A time table has many fields to store information about its genetics, costs, number of violations of constraints. The timetable for an entire university is therefore a collection of room timetables,

one for every room in the university as shown in Figure 6.2. A population is a collection of timetables, which also have many fields to store information about timetables like less costly timetable, most costly timetable, average cost, average number of violations of constraints and the total number of timetables in population. It contains a pointer to the least costly timetable in the population, (which has, in turn, a pointer to the next least costly). Timetables are ordered from least costly to most costly. A colony of creatures is therefore a singly-linked list of structured types (creatures) containing timetable data (genes) in a three dimensional array. Figure 6.3 shows the way in which a population is comprised of a linked list of timetables.

	Mon	Tues	Wed	Thu	Fri
8:30	322 AQ	422 EN	513 MECH	228 PHY	0
9:30	558 SO	515 HIS	259 FR	0	332 PR
10:30	…	….	….	…..	……..
11:30					
12:30					

Figure 6.1: Example of a single room timetable

A university timetable stores information about what classes are booked in each room, at any hour of the day, on any day of the week. Each of these bookings (or NULL bookings) is one gene. A timetable also has fields which describe (decode) some aspect of this genetic information. A timetable has a field which stores its cost. It also has fields which store the number of breaches of each type of hard constraint.

A two stage verification strategy is used which ensures that each lesson of a course is scheduled exactly once. It is done in two steps, in first step checking each lesson which appears more than once altered in such a way that it appears exactly once and in second step any lesson which did not appear is booked to spare spaces randomly. The benefit of this representation is that room must not be double booked and every lesson must be scheduled at

once. In this chapter, we will explain in detail our algorithm for solving educational timetabling problem.

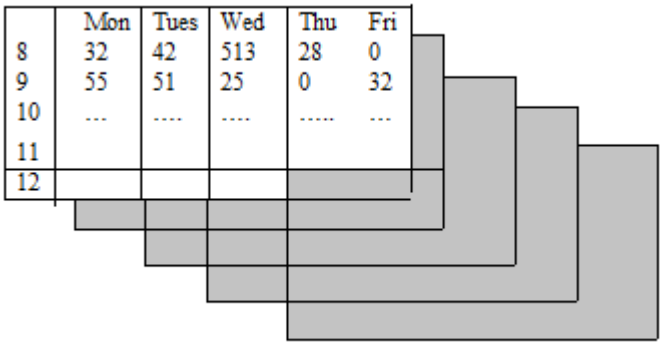


Figure 6.2: An entire university timetable

6.1.1 Reduction of search space

Now we shall discuss about our chromosome representation, search space issues and benefits to use this representation. We are using two-dimensional matrix (i.e. grid) representation for our chromosomes where each cell in this matrix is representing an empty slot or at most one event .Benefit of using this representation is that it reduced the search space significantly. If there are *m* events to schedule in *t* places then the total number of possible ways to schedule *m* events in *t* places is t^m where *t* places means that total available slots which are total number of rooms multiplied by total number of periods (if rooms and periods are always available). Now the claim of reduction of search space by using the method of this encoding can be proved in this way.

The number of ways to assign *m* events to *t* places as we have used in our representation are $\frac{t!}{(t-m)!}$.

Now this $\frac{t!}{(t-m)!}$ relation can further be elaborate in this way,

$$\frac{t!}{(t-m)!} = \frac{t\times(t-1)\times(t-2)\times.....\times(t-(m-1))(t-m)(t-(m+1))\times.....\times 2\times 1}{(t-m)\times(t-(m+1))\times.....2\times 1}$$

$$= t \times (t-1) \times (t-2).....(\ t-(m-1)) \qquad \text{(A)}$$

Now comparing t^m with (A), one can prove the above claim.

$t^m = t \times t \times .....m(time).....\times t$ which is always greater than (A).

$t^m = t \times t \times .....m(time).....\times t > t \times (t-1) \times (t-2).....(t-(m-1))$.

Where *t* is greater than *m* or equal to *m* always ($t \geq m$) .Because this is a necessary condition to get feasible solution otherwise feasible solution could not be achieved.

One benefit of this presentation is that double room booking clash can be simply finished. Double room booking means that to schedule more than one lesson in a room in a period. So by using this method on encoding one can avoid one important hard constraint.

We have tackled many other complication steps by using this representation which are helpful to speed up each of remaining procedures of algorithm. These steps involved some additional matrices.

6.1.2 Benefits of using this representation

By making use of a two-dimensional matrix representation in this thesis, now we shall explain how many matrices can be made by using this representation.

Event-Room Matrix

This matrix is used to indicate which event is suitable for which room. This is a Boolean matrix and can easily be calculated that which room r satisfies the conditions to host event e. Thus if room r satisfies the conditions to host event e then the element (e,r) marks as true otherwise it marks as false.

Conflicts Matrix

This matrix is very similar to adjacency matrix used for representing graphs. For our problem it indicates which pair of events have conflicts (so can not be scheduled in the same period). For example if any two events e_1, e_2 have any conflict then element (e_1, e_2) in the matrix marks as true otherwise marks as false.

By using this encoding counting of violations is easy and inexpensive. To check violations are easy now, for example if one wants to check that each class should have at most one lesson in any period, it can be evaluated by checking each column whether this column is true more than one entry or not (rows of the matrix are classes). If one wants to check that proper room type *r* has been assigned to an event *e* in timetable, simply it can be verified by checking entry (*e*,*r*) is true in event-room matrix.

We have suggested some more matrices for our problem after following the suggestions of Carter [129] (Chapter 1, Section 1.7).

Room- Period Matrix

This matrix indicates that room *r* is available at period *p* or not. If room *r* is available at period *p* then element (*p*,*r*)marks as true otherwise it marks as false because in our problem rooms are not always available.

Course-Period Matrix

This matrix shows the relationship between period and events because some events can not be scheduled in some periods. If event *e* can be scheduled in period *p* then element (*e*, *p*) will be marked true otherwise it will be marked false.

Class-Period Matrix

This matrix indicates class period relationships. One benefit of this matrix is to identify the position of class with respect to sessions. Because sometimes some classes should be scheduled in morning sessions and some should be scheduled in evening sessions. If class *k* can be assigned in period *p* then element (*k*,*p*) marks as true otherwise it marks as false.

Period-Teacher Matrix

This matrix would relate period and teacher requirement. Because some teachers are not available on certain periods so this requirement can be indicated by this matrix. If teacher *t* is available at period *p* then element (*p*,*t*)marks as true otherwise it marks as false. One can notice that this matrix is not equivalent to course-period matrix. For example, if a course is not available on a period then this does not mean that teacher of that course is not available. It is possible that teacher is available on that specific period to teach any other course.

6.2 Initial solution

Population based algorithms are generally used with an initial population. This initial population sometimes is generated randomly and sometimes by using special techniques to make a higher quality initial solutions for population. These techniques are used to give the algorithm a good start and speed up the process. For memetic algorithm, we have generated our initial solution randomly and then repaired it by our proposed repair strategies. For honey bee mating algorithm, we have used heuristics to make initial population of drones and queen.

6.2.1 Pre-processing or division of search space

1. Sets of Lessons

Lessons take place during working days. Every lesson has the following information: course, class, teacher, size type, where course means the name of course; class means name of the group of students taking this course; size is the number of students taking this course; teacher means the person who teaches this course and type indicates the kind of subject, i.e.; lesson, experiment etc. This type of lesson decides which type of room is required for this event.
We shall divide the lessons into sets on the basis of their common properties to use them for proposed heuristics.

Population Size	2
Average Cost	323
Violation of hard constraints	150
Pointer to best timetable	Pointer to worst timetable

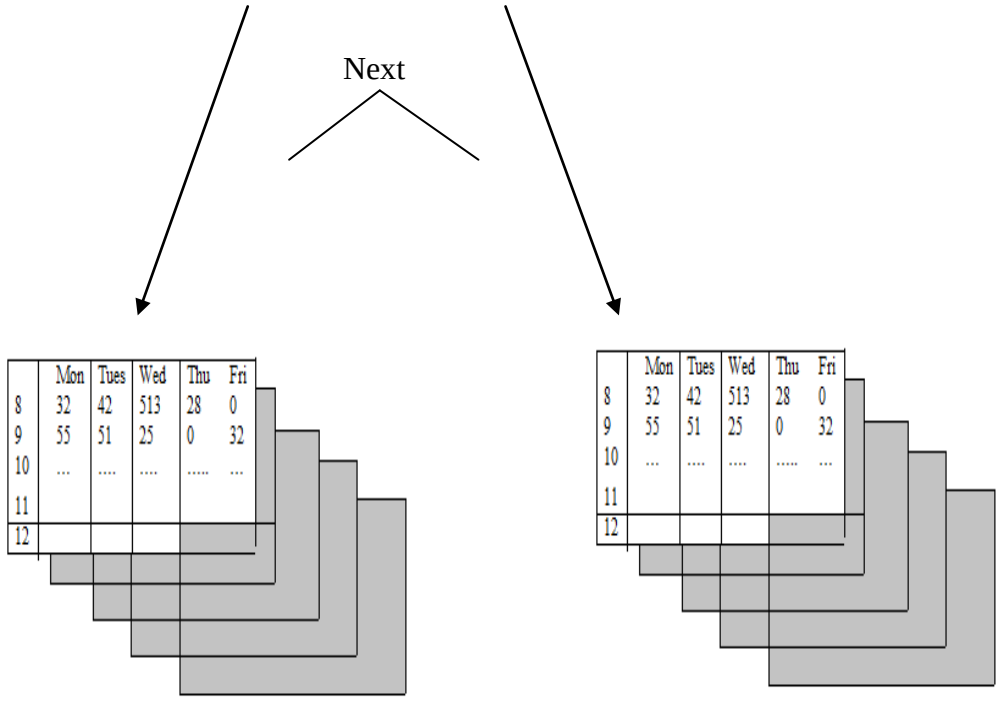


Figure 6.3: Population of timetables

2. Sets of rooms

All the rooms are divided into sets. Now each set of rooms has some common properties, for example lecture rooms, laboratories, large capacity rooms, rooms with projectors and rooms which can be partitioned into more rooms. Now each set of room is allocated a type of room and each room will be distinguish according to its type. Each type of room has its own

comprehensive meanings. Then each set of rooms (according to its type) is ordered by its capacity. This capacity ordering would be used in a heuristic in later stages. Each room has the following information: name, capacity and type. Where name indicates the name of room; capacity means the number of seats in the room and type means the kind of room, i.e.; lecture room, laboratory etc.

These three types of procedures have been used to construct initial solutions for initial population of our algorithms.

1. Random initialization
2. Repair strategies
3. Set forming heuristics

We describe these methods one by one in detail.

6.2.2 Random initialization

Individual solutions can be generated randomly to form an initial population. The size of population depends on the problem. It can contain several hundreds or thousands of possible solutions depending upon the problem nature. For our problem it varies from 50 to 100 because our preliminary experiments have found this range good results producing. The benefit of this strategy is that there would be a large diversity in solutions of search space.

6.2.3 Repair strategies

After getting the initial solution repairing is mainly done on violation of hard constraints. First of all know about the location of the offending slots and replace them iteratively with valid slots.
In our algorithm we have used room repair strategy and two step verification strategy which are given in Figure 6.4 and Figure 6.5.

In room repair strategy, we choose an assigned course which causes hard constraint violation of rooms. We delete this assignment and reassign it to another slot in a way that no violation of room related constraint occurs. The procedure for this strategy is explained in Fig.6. 4.

```
For every lesson assigned
If (there is any room related hard constraint violation)
Remove this assignment
For all rooms
If a room is available at period P
Find a room for which (Size of class<= capacity of room)
Assign the lesson to that room-period pair
Endif
End for
```

Figure 6.4: Procedure for room repair strategy

Two step verification strategy

We can call our two step verification strategy as Guided Search strategy or guided two step verification strategy or heuristic.

Stage1

```
Repeat for every course Ci
   Check If  Num_ assigned > Numlects
    while (course_occurence = Numlects) do
     randomly remove an assignment
   endwhile
```

Stage2

```
Repeat for every course Ci
Check If Num_ assigned < Numlects
     while (course_occurence = Numlects) do
          Choose a room-time slot pair randomly
         Assigne this cours
     endwhile
```

Figure 6.5: Two step verification strategy (Repair procedure for violation of two hard constraints)

In two step verification strategy, we fulfill two hard constraints.

(1) Every course should be assigned.

(2) The number of assignments of every course should be equal to its number of lessons.

In first step we see if a course is assigned more than its number of lessons, we remove the extra assignments. In second step we see that if any course is missing or it has assigned less than its number of lessons, we find a new slot randomly and assign it. At the end of this two step verification strategy every course is assigned and its number of assignments is equal to its number of lessons. This is explained in Figure 6.5.

6.2.4 Set forming heuristics

We use the set forming heuristics to make initial solutions for population.

(H1) Make a list of courses in descending order with respect to the strength of course. Strength of course means that how many students take this course.

(H2) Make a list of courses in ascending order according to their room availability. The course with less available rooms comes first in list.

(H3) Make a list of courses which have more lessons to schedule. This list is also in descending order.
We use these two heuristics (H4 and H5) for the assignment of slot to selected lesson.

(H4) We shall assign slot to these courses randomly.

(H5) We make a list of rooms in ascending order of their capacity.

Then we assign rooms to courses (already selected by a heuristic). For this assignment, we first choose the first room of our list. If the capacity of first room is greater or equal to the size of course, we assign this room to the course. Otherwise we move down in the list until we get the room of required capacity. By doing so, we can save the big capacity rooms for the larger classes in the later stage. In this way, the room capacity conflict can be reduced.
A lesson is chosen according to heuristic H2 and ties were broken by H1 and further ties with H3. Then to find a place for the lesson heuristic H5 was used and ties were broken by H4.

6.3 Memetic Algorithm

In this section, we propose our memetic algorithm. We discuss its different operators, initial

solution method and local search which is integrated to improve the performance of the algorithm.

6.3.1 Classic genetic algorithm

First of all, algorithm initializes a population of individuals and evaluates the fitness of each member in that population. The procedure for applying these genetic operators is given as:

- Select the best-fit members for reproduction
- Breed new members through crossover and mutation operations to give birth to children
- Evaluate the individual fitness of children
- Replace least-fit members of population with children

This process repeats until the termination criterion is achieved.

6.3.2 Genetic operators

Evolutionary algorithms are good tools for a big space optimization problems but their performance depends a lot on the type of genetic operator used and the values of parameters such as mutation rate, crossover rate and population size. Mutation operators normally do the random alteration of genes. This is done during the process of copying a chromosome from one generation to the next. Crossover operator is used for exploration in these algorithms.
We used steady state algorithm and general memetic algorithm discussed in Chapter 1, Section 1.8 in our preliminary studies and then compared them. We found general memetic algorithm more efficient as compared to steady state algorithm in our case.

Now we describe the three genetic operators and their types in detail.

1. Selection.
2. Cross over
3. Mutation

6.3.2.1 Selection

We have used these selection operators: elitism, roulette wheel selection and selecting one parent by elitism and the other parent by roulette wheel selection, after selecting parents, we apply breeding and mutation.

6.3.2.2 Cross over

We have used different types of crossover operators in our preliminary experiments for the breeding of two parents for proposed memetic algorithm.

1. one-point Crossover
2. two-point Crossover
3. Uniform crossover
4. Apply crossover to some percentage of population and not to all.
5. No crossover at all.

The details of one-point crossover, two-point crossover and uniform crossover are given in Chapter 1, Section 1.8. We have used these crossover operators for our preliminary experiments but uniform crossover is performing better for our problem. For this purpose we generate a crossover mask randomly for each slot. In this way the child contains a mixture of genes from each parent and the procedure is explained in Figure 6.6.

```
Select two parents P1, P2. [Based on the Selection operator chosen];
For every slot X1
Generate random mask bits {0,1} for the selected slot
if (mask bit equals 1)
Find the information for the slot X1 in P1;
else if (mask bit equals 0)
Find the information for the slot X1 in P2;
end if
```

Figure 6.6: Procedure for crossover

6.3.2.3 Mutation

Mutation is used to avoid from getting trapped on local optima. A mutation operator is used with a mutation probability *Pm*. *Pm* is the probability that a gene will undergo mutation and its value is 2 * Mutation rate /1000.

We have used two types of mutation for our algorithms. For memtic algorithm, we have used random mutation and for honey bee algorithm we have applied mutation by neighbourhood structures. These are explained in the following:

(i) We have applied random mutation. The procedure for this mutation is explained in Figure 6.7.

(ii) Many neighbourhood structures were tried within the iterative improvement algorithms; these were tested on post enrolment course timetabling problems (ITC 2007). This proposed approach found some of the good results. Based on their experiments, we have applied some neighbourhood structures within our algorithm. The detail of these neighbourhood structures is given in (Chapter 1, Section 1.6).

A mutation operator is used with a probability Pm. The mutation operator first randomly selects one from two neighbourhood structures N1 and N2 described in Chapter 1 in Section 1.6 (in Chapter 1, they are named N3, N10), and then makes a move within the selected neighbourhood structure. The procedure is given in Figure 6.8.

```
For each chromosome
For every gene of that chromosome
Generate a random number r by U(0,1000).
If (r<Pm)
Swap current gene with a randomly chosen gene from the current timetable
else
keep the gene un-mutated.
endif
Repair the timetable using two step verification
```

Figure 6.7: Procedure for random mutation

The neighbourhood structures are described as follows:

N1: Neighbourhood defined by an operator that an event can be moved only if the corresponding slot is empty.

N2: Neighbourhood defined by an operator that swaps the slots of two events.

We can apply mutation to some percentage of population and not to all. For this we can do mutation on 10% or 20 % of the members of the population. And for each member, we can mutate 1 to 20% of the lessons. Lessons are chosen at random from any slot and are reallocated to the next earliest possible slot.

Based on our preliminary tests, we observed that these percentage choices did not show any significant effect on mutation procedure so we did not implement them in our algorithm.

For each chromosome
For every gene of that chromosome
Generate a random number *r* by U(0,1000).
If (*r<Pm)*
Swap current gene with a gene chosen in the neighbourhood N1 or N2 from the current timetable
else
keep the gene un-mutated.
endif
Repair the timetable using Two step Verification

Figure 6.8: Procedure for mutation by neighbourhood

6.3.3 Infection

For memetic algorithm, we have used an infection in population at the chromosomes level instead of genes. It does not work in a deterministic way but it occurs randomly. For this we generate a parameter called number of infections by uniform distribution in (0, 10). These infections will take place after a random number of generations g_{inf} . The value of g_{inf} can be between 0 and maximum number of generations. By doing so, we kill a portion of randomly chosen timetables (m_inf) of the population colony. Here we have taken m_inf equal to 10 % of the population. And then we restart memetic algorithm by exploiting infection results. The procedure is given in Figure 6.9.

```
Pseudo Code for infection
Find out number of timetables to infect m_inf
Select m_inf timetables randomly
Kill these timetables
Restart MA
```

Figure 6.9: Infection

6.3.4 Replacement

After creating new children by using crossover and mutation operators a successor generation is made. As parents chromosomes are selected according to their fitness value, so it is hoped that the children go towards the good fitness value after each generation. This is the replacement procedure which decides about child survival or extinction.

The following replacement methods have been used in the literature: random replacement, replacement with elitism, replacement with worse population and steady-state replacement.
In random replacement, some specific percentage of members is randomly chosen in the population and replaced with new members. This type of replacement provides us with enough randomness to have diversity in search space but on the other hand we can loose some good solutions. In steady state replacement, n old members are selected in the population and replaced with n new members. The choice of the number n and the decision of extinction of chromosomes from the current population are important aspects of the genetic algorithm. It is found that steady state enhances the speed of algorithm. The reason is that one changes only one or two members of population at each generation while the other population remains the same. In replacement with worse population, one can replace the worse chromosomes of the population and apply crossover and mutation operators on those worse chromosomes. The better chromosomes are directly copied to the next generation.
Since the aim of algorithm is to improve the objective function value in each generation, we have used replacement with elitism. We keep best members of the population and delete all other ones. By this way, good solutions are preventing from becoming extinct.

6.3.5 Proposed Local search

The local search (LS) techniques in genetic algorithms are used for improving the quality of timetables. In our case, we take one or two courses randomly, and swap them by using our neighbourhoods defined below. If it improves the objective function, we accept it otherwise we reject it. If the fitness is adjusted, the chromosome which is the result of local search will be replaced with the primary chromosome; otherwise that primary chromosome is identified as the best solution in its neighbourhood and will remain unchanged. We have used these neighbourhoods for our local search. The pseudo code for the Local Search is described in Figure 6.10.

The procedure of Local Search (LS)

1: **input:** Start with a solution s_0 selected from the population

2: **while** Termination condition not reached or minimum cost not achieved **do**

3: **if** cost of solution s_0 is greater than minimum cost defined **then**

4**: for** i = 1 to n **do**

5**:** examine the solution ***s*** in the neighbourhoods of s_0 (first by N1and then N2)

6: **if** the cost of solution ***s*** < the cost of solution s_0 t**hen**

7: Set s_0 =***s*** and register if ***s*** is the best solution found so far.

8: **end for**

9: **end if**

10: **end while**

11: **output:** A possibly improved individual **I**

Figure 6.10: The pseudo code for Local Search

N1: Neighbourhood defined by an operator that an event can be moved only if the corresponding slot is empty.

N2: Neighbourhood defined by an operator that swaps the slots of two events.

We can use local search in the following two ways.

a) After making an initial solution, we apply LS to every member created and then put it to population pool.

b) After applying GA, apply LS to child before putting it to population pool.

We applied these two ways but in (a), the effect of LS is smaller as it applies only to initial solution while in (b), LS improves each generation and improves the quality of solution. So we have used (b) in our algorithms.

6.3.6 Pseudo code of memetic algorithm

In a conventional memetic algorithm, a hill climbing method is applied after the mutation. We propose a memetic algorithm that integrates a local search into the genetic algorithm for solving the university timetabling problem. This local search method uses its exploitive search ability to improve the explorative search ability of genetic algorithm. The pseudo code for proposed memetic algorithm is shown in Figure 6.11.

1. **Algorithm 1** Pseudo code for Memetic Algorithm (MA)
2. input : A problem instance I
3. set the generation counter g := 0
4. **while** (solution_colony population_size < **n) do**
5. create a timetable t_i by random initialization method
6. repair this timetable t_i by proposed repair strategies
7. calculate the cost of t_i
8. enter t_i to the population colony
9. **end while**
10. **while** the termination condition is not reached **do**
11. replace 20 % members of the colony
12. **while** (solution_colony. population_size < **n) do**
13. choose two parents via roulete wheel selection
14. S_i child solution generated by applying the uniform crossover operator with a probability p_x
15. S_i child solution after mutation with a probability p_m
16. Calculate the cost of S_i
17. S_i' improved child solution after applying proposed local search on S_i
18. If cost of S_i' is less than cost of S_i , accept S_i' otherwise choose S_i
19. enter this timetable to the population colony
20. **end while**
21. g := g + 1
22. after g_{inf} generations, apply infection to the population colony
23. **end while**
24. **output** : The best achieved solution S_{best} for the problem instance **I**

Figure 6.11: The pseudo code for memetic algorithm

6.4 Honey bee algorithm

There are two major types of Honey bee algorithms. First type is a honey bee colony algorithm which was mostly used to solve educational timetabling problems, second type is honey bee mating algorithm.

6.4.1 Honey bee colony Algorithm

Honey bee colony algorithm is about the food collection of the bees. It is a nature inspired algorithm. A colony of honey bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources. A colony will be developed by deploying its foragers to good fields. The patches of flowers with plentiful amounts of nectar or pollen that can be collected with less effort will receive more bees and patches of flowers with less nectar or pollen should be visited by fewer bees.

The foraging process starts in a colony by scout bees being sent to search for promising flower patches. A colony employs a percentage of population as scout bees for exploration of the fields and these scout bees go randomly from one patch to another, when these scout bees come back in hive; they share information with other bees. This information helps the colony to send its bees to flower patches precisely. These information make the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it. Then these scout bees go back to flower patches with follower bees that were waiting inside the hive. More follower bees are sent to more promising fields. This helps the colony to gather food quickly and efficiently. The bees monitor food level of the flower patch, if patch is not good enough bees go back in hive and if the patch is still good enough as a food source, then it is advertised and more bees are recruited to that field.

6.4.2 Honey bee mating algorithm

In honey-bee colony, there is a queen(s) (best solution), drones (incumbent solutions), worker(s) (heuristic) and broods (trial solutions). The algorithm describes the natural mating behavior of the queen bee when she goes for mating with drones. Queen adds sperms of drones in her spermatheca during her visit after each successful mating. Queen spermatheca has a fixed size and when this fills up queen comes back in hive to start mating process. After mating process broods are produced which are then fed by worker (all new generated broods

are grown up by a worker).The number of workers used for algorithm means that the number of used heuristics. If the fittest brood is better than queen then it replaces her.

The strength of this algorithm is to explore simultaneously and exploit problem search space. This is achieved by the queen's transition in the search space and employing a local search at each iteration. The queen (current fittest solution) is the best individual, so it is hoped that it will evolve superior solutions. This dominated solution stores different drone's genotypes in her mating pool. Some parts of these genotypes are used to make new broods by combining some parts of the drone's genotype with parts of the queen genotypes. These features of honey bee algorithms make it different from other population based algorithms those have been used for educational timetabling problems.

6.4.3 Proposed honey bee mating algorithm

We have proposed a honey bee mating optimization algorithm for educational timetabling problem. According to our knowledge, there are few articles on educational timetabling problems which use honey bee mating algorithm (Chapter 1, Section 1.8).

The honey-bee mating algorithm was used first time to solve educational timetabling problem by Sabar et al. [48]. They referred it as HBMO-ETP algorithm. In this work, we propose a variation of HBMO-ETP, which we call as HBM algorithm. The pseudo-code for our proposed HBM algorithm is shown in Figure 6.12. Now we describe our algorithm in detail.
Firstly, we select a number of honey-bees to create the population of the initial hive. In many cases, random generation methods may not necessarily guarantee a good quality solution. Therefore, in this work, we employ heuristics which are described earlier in this chapter to make an initial population of drones.

At this stage, the solutions generated may or may not be feasible. For the moment we do not impose the feasibility condition in order to maintain the diversity. We choose the fittest solution in this population and make it queen. The other solutions of this initialization phase become the drones.

Approa-ches	Application	Solution represent-ation	Same representati-on as HBMO - ETP	Initializati-on method	Initial solution	Crosso-ver	Mutation	Probl-ems	Exploitation during search
Our HBM algorithm	improvement	direct	no	heuristics	infeasible	uniform	neighbourhood structures	exam and course	yes
HBMO-ETP [48]	improvement	direct	_	graph colouring	feasible	haploid	shaking procedure	exam and course	yes
GA [53]	constructive + improvement	direct	no	random	infeasible	local search	local search	exam	no
ACO[54]	constructive	direct	no	graph colouring	_	_	_	exam	no
GA [56]	improvement	direct	no	graph coloring	infeasible	_	move operator	exam	no
GA [57]	improvement	indirect	no	graph colouring	feasible	one point	_	exam	no
GA [52]	constructive + improvement	direct	no	GA	feasible	one point	move operator	exam	no
ACO[51]	constructive	direct	no	_	feasible	_	_	course	no
GA [59]	improvement	direct	no	random + graph colouring	feasible	_	move operator	course	yes
GA [101]	improvement	direct	yes	random	feasible	one point	move operator	course	yes
GA+ GD [58]	improvement	direct	no	graph colouring	feasible	_	Random swap	course	yes
EM+ GD [60]	improvement	direct	yes	graph colouring	feasible	_	_	course	yes

Table 6.1: Differences and similarities between our HBM algorithm and previous population based algorithms. ‘‘–’’ means the method did not use the corresponding operator.

The solutions in the population are ranked according to the probability value p_i.This can be calculated using equation (1).

$$p_i = \frac{f_i}{\sum_{i=1}^{NS} f_i} \qquad (1)$$

Where *NS* = total number of solutions in population i.e. population size, f_i = fitness value of the *i*th solution. Next, we divide these drones into two categories: (1) Common drones and (2) Elite drones, on the basis of their probability values. Here we divide the drones in a way that the elite drones have more chances to be improved as these solutions are considered the most promising solutions in the search space. In HBMO-ETP, they consider that all the drones are of the same type.

1. Set the number of queens=1, iter =0
2. Set total no. of workers W; queen spermatheca is empty $|q_s|$ = ϕ
3. Set no. of workers for common drones W_c
4. Set no. of workers for elite drones W_e
5. Set the maximum size of queen spermatheca= $q_{s-\max}$
6. Set no. of common drones, n_{cd} =0; maximum no. of common drones $n_{cd-\max}$
7. Set no. of elite drones, n_{ed} =0; maximum no. of elite drones $n_{ed-\max}$
8. Set no. of broods n_{br} =0;
9. For *i* =1 to $|n_{cd-\max} + n_{ed-\max} + 1|$
10. Use the heuristics to generate the drone solution S_i and add it to drone population
11. End For
12. While iter $\leq$ M
13. iter= iter + 1;
14. Calculate the fitness value of each drone
15. Select the fittest drone and set it to as queen *Q*
16. Calculate the probability value p_i by using equation (1)
17. Divide all drones in two categories: (1) common drones and (2) elite drones
18. Select $n_{ed-\max}$ elite drones and $n_{cd-\max}$ common drones on the basis of p_i
19. While $|q_s|$ < $q_{s-\max}$
20. Select Nc-mat drones from the set of common drones via roulette wheel selection
21. Select Ne-mat drones from the set of elite drones in a deterministic order
22. End while
23. For *j*= 1 to $|q_s|$ do
24. For *i*=1 to Nc-mat
25. Create a brood B_c by mating queen *Q* and common drone sperm via uniform cross over
26. Apply a worker selected from W_c to grow brood B_c i.e. to improve its fitness value
27. End For
28. For *i*=1 to Ne-mat
29. Create a brood B_e by mating queen *Q* and elite drone sperm via uniform cross over
30. Apply a worker selected from W_e to grow brood B_e i.e. to improve its fitness value
31. End For

32. If the fitness value of any of the brood created is $f(S_j^*)$ better than Queen's $f(Q)$
33. Replace the queen with that brood
34. $Q = S_j^*$ and $f(Q) = f(S_j^*)$
35. Else add S_j^* to population
36. End For
37. Mutate all broods by using random neighbourhood structures (N1 and N2)
38. Kill the old drones and insert the new mutated broods into population
39. End While
40. Return the queen (Fittest solution found)

Figure 6.12: Pseudo Code for Honey Bee Mating Algorithm

In lines 19-22 (see Figure. 6.12), we describe the mating flights of the queen. In each mating flight, queen selects Nc-mat drones from the set of common drones and Ne-mat drones from the set of elite drones. Here we have eliminated the speed and energy parameters which were used in the original HBMO algorithm [68]. Instead we have selected drones via Roulette wheel selection and in a deterministic way. By doing so, we shall exploit the probabilistic nature of RWS to select more diverse solutions. In HBMO-ETP, they eliminated speed parameter but maintained energy parameter to initialize queen's energy for mating flights. This energy parameter defines the number of drones selected during a mating flight. While we have eliminated the speed and energy parameters and used the following two parameters Nc-mat (number of common drones for mating during a flight) and Ne-mat (number of elite drones for mating during a flight). If the mating is successful (according to the probabilistic decision rule), the drone's sperm is added into the queen's spermatheca. This process continues until the queen spermatheca is filled.

Next, the queen starts breeding and two types of broods are formed by mating with common drones and elite drones via uniform cross over. While in HBMO-ETP, they applied haploid cross over. The reason for using uniform cross over is to explore the whole solution space and create more diverse broods. After that, the workers already defined for elite broods are applied (i.e. an improvement) to elite broods and the workers specific for the common broods are applied to common broods. We utilize different neighbourhood structures as the workers to grow the broods i.e. to improve the trial solutions. As in the original honey-bee mating optimization algorithm, the workers improve the brood produced from the breeding queen

with the possibility of replacing the queen if the improved brood is better than the current queen.

Table 6.2 shows a summary of differences and similarities between our HBM and that of original algorithm [68] and its variant HBMO-ETP.

Parameters of algorithm	HBMO by Abbass [68]	HBMO-ETP [48]	Our proposed HBM
Drones generation	random	LS+LD+LE	heuristics
No. of queens	1	1	1
Drones selection dependence	energy+speed	energy	RWS + deterministic (no parameters of energy and speed)
Local search	greedy SAT	simple descent	hybridized local search
Crossover	haploid	haploid	uniform cross over
Mutation	flip	shaking procedure	Various neighbourhood structures
Resultant broods	all broods are killed	all broods are used in the next mating flight	broods will replace the drones
Fitness function	fitness function	objective function (timetable quality or penalty cost)	objective function (weighted sum of penalties for soft constraints)

Table 6.2: Differences and similarities between our HBM algorithm, HBMO [68] and HBMO-ETP [48]

In HBM algorithm, we update the population of drones iteratively to avoid undeveloped convergence. In the original HBMO [68] all broods are killed and the new mating flight begins using the previous population. But we kill the older drones used in breeding process and replace them with the new mutated broods. So the next mating flight starts with fresh drones. This replacement guarantees that each drone's sperm can be used only one time which, we hope, helps in maintaining diversity and prevents immature convergence. During each mating flight, queen selects one elite drone from the set of elite drones in such a way that

each elite drone will be selected at the end of mating flights. And queen chooses Nc-mat common drones via RWS from the set of common drones during each mating flight.

Lines 1–8 show the initialized values for the defined parameters. These are following: (i) the number of queens (ii) the queen's spermatheca size, which represents the maximum number of mating each queen performs in a single mating flight, thus also the number of broods that will be born after each single mating flight and (iii) the number of workers to improve common broods and elite broods. We use workers (heuristic Search) for our common broods and elite broods.

In lines 9 to 11, the drone population is generated by using heuristics described earlier. The status of queen Q is given to the fittest solution in lines 14 and 15. Lines 16-18 show the division of drones into two categories on the basis of their probability calculated using equation (1). In lines 19-22, we describe the mating flights of the queen. In each mating flight, queen selects Nc-mat drones from the set of common drones and Ne-mat drones from the set of elite drones. The accepted drones are added to the queen's spermatheca. Based on Eq. (1), the fitter drones have more chance of being selected. This procedure is repeated until the queen's maximum spermatheca size is reached.

Then, the breeding process starts from line 23. Two types of new broods are created. In lines 24–28 we use Uniform cross over for mating of queen and common drone. Then an already specified worker (Local search hybridized with heuristic) is recruited to grow this brood. In lines 29–32 queen mates with an elite drone by uniform cross over to produce another brood. Then a worker specific for elite broods (Local search hybridized with heuristic) is recruited to grow this brood. The mating generates a new set of broods.

This hybridized local search starts with an initial solution (brood) and iteratively improves it by examining its neighbourhood. A neighbourhood of a given solution is obtained by moving one event from its current slot to another slot, which is selected by a heuristic. The solution is accepted, if the move does not violate any hard constraints and the quality of the neighbourhood solution is better than the incumbent solution. Otherwise, the solution is rejected and a new event is selected to generate a neighbourhood solution.

In our HBM algorithm, a chromosome is used to represent a candidate solution S_i to the problem. We have used the same representation of chromosomes as we employed for our memetic algorithm. If the improved brood is better than the queen, the queen is replaced by the brood. Otherwise we keep the original queen as the best solution. The new broods will be modified using a mutation operator. In this work, we employed two neighbourhood structures (N1 and N2) for this purpose. These neighbourhoods are applied to each brood. The modified broods will replace the older drones for the next mating flight. This process is repeated until the stopping condition is satisfied. It may be time limit, number of iterations or allowed minimum value for objective function. In this algorithm, we take the time limit as stopping condition. As the time limit is reached, the fittest solution found is returned as queen. And this final queen is the solution of the given problem.

Parameters	Value
Population size	100
Number of queens	1
Number of drones	99
Number of mating flights	4
Size of queen's spermatheca	5
Number of elite drones	4
Number of common drones	95
Mutation rate	7
Killing ratio	0.2
Ne-mat	1
Nc-mat	4

Table 6.3: Parameter settings for our HBM algorithm's computational experiments

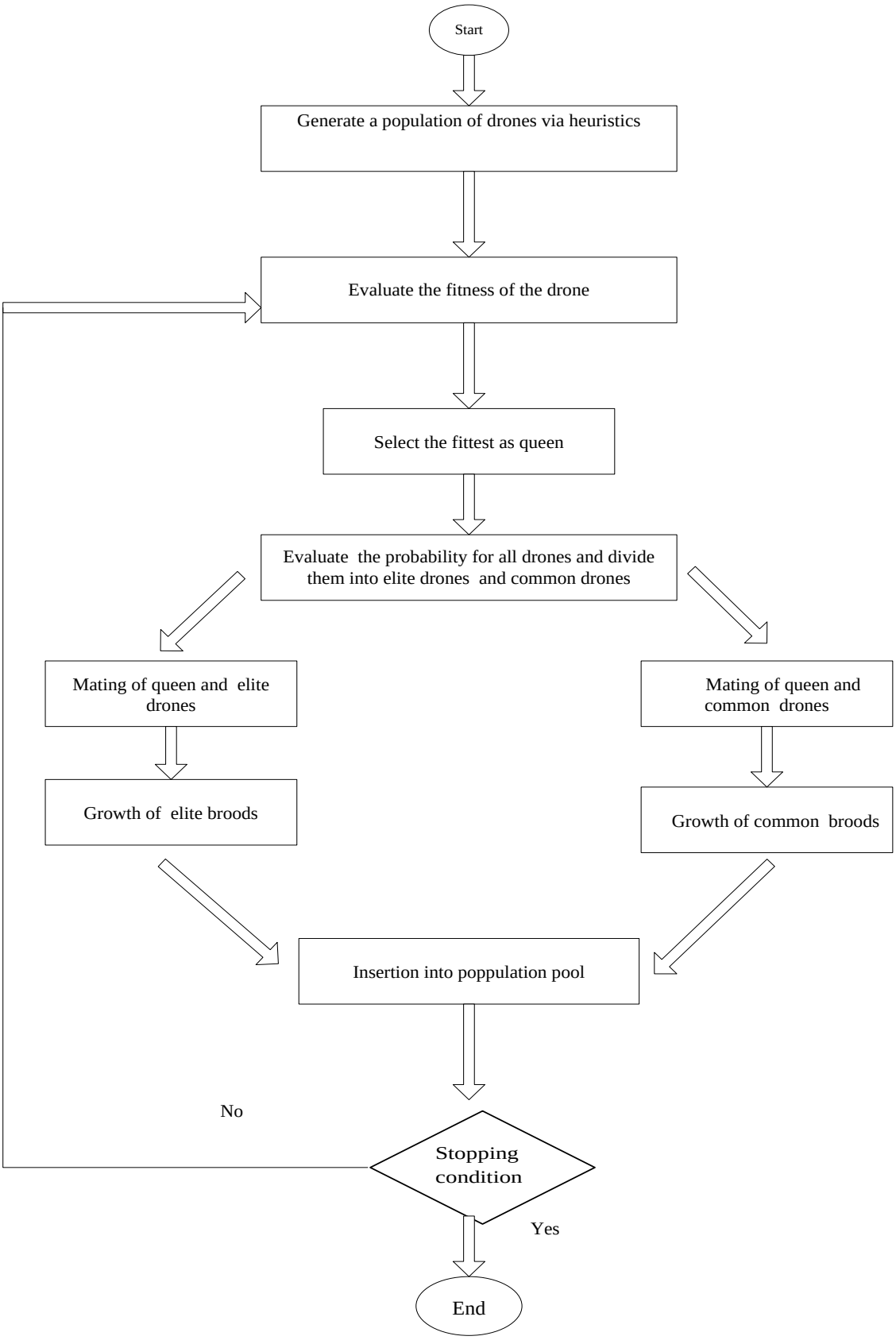


Figure 6.13: The block diagram for the proposed algorithm

Figure 6.13 shows the block diagram for the proposed algorithm on university timetabling problems. It illustrates the process of our proposed algorithm. The algorithm starts by generating an initial solution via heuristics. Next, the improvement process (honey bee mating algorithm) is executed (as discussed earlier). Table 6.3 represents the parameter settings for our HBM algorithm's computational experiments.

6.5 Termination criteria

This iterative process continues until one of the possible termination criteria is reached. Different types of termination criteria can be used such as, getting optimal solution (in case of some easy datasets), obtaining acceptable solution, maximum number of generations, time limit and no improvement in objective function value during a certain number of generations. We have used time limit or getting optimal solution as termination criteria for our problem which achieves first.

6.6 Conclusion

In this chapter we have proposed our algorithms for solving university course and examination timetabling problems. In any algorithm its solution representation plays a significant role for making its performance good. We have discussed in detail chromosome representation of our memetic algorithm. This matrix form representation has many benefits to evaluate violations of constraints and in reducing the search space. We have defined many matrices from this matrix representation which helps the algorithm for finding violations of constraints like room- period matrix, course- period matrix, class- period matrix, period-teacher matrix, event-room matrix and conflict matrix.

Our algorithms are population based algorithms. We have discussed to make initial solutions for population of these algorithms. We have explained in detail the procedure of making initial solutions for algorithms. We have used many ways to make initial solutions in our prerequisite experiments and have finally explained which procedures are producing good results for our experimental work. We have demonstrated the procedure of making initial solution by our proposed heuristics. We have also talked about the procedure of our cross over and mutation for memetic algorithm. How much different selection ways have been used

for parent's selection for breeding have been presented and which way is producing good results for our algorithm have been mentioned.

Creation of new children by using crossover and mutation operators succeed towards a new generation. Thus it is important to select some population members, which will be replaced with new children. We have mentioned our replacement procedure. We have explained the procedure of our local search, its neighbourhood structures and the procedure of our proposed memetic algorithm which uses this local search for the improvement of the solution quality.

We have also proposed honey bee mating algorithm. This describes the natural mating behavior of the queen bee when she goes for the search of drones with her spermatheca which has a fixed capacity and when this fills up queen comes back in hive to start mating process. Mating process produces broods fed by worker and finally if new fittest brood is better than queen, it replaces her. Initial population of hive is created by our heuristics. We have used probability function to rank population and to choose queen for hive. We have divided these drones into two categories common drones and elite drones on the basis of their probability values. We have described differences and similarities between our HBM algorithm and previous population based algorithms in detail.

Chapter 7

7 EXPIREMENTS AND RESULTS

In this chapter we have analysed results obtained by our proposed algorithms. We have discussed in detail reasons of obtaining good results and also shortcomings of the algorithms. We have applied our algorithms on one benchmark timetabling problem, generelized educational timetabling problem and generalized examination timetabling problem. First of all we have applied our memetic algorithm on benchmark timetabling problem (Curriculum based course timetabling problem). This problem is a part of international timetabling competition problem 2007. This problem is discussed in detail in (Chapter 5, Section 5.1.2). We have compared our results on benchmark instance with other algorithms used in literature.

Then we have generated data for both generalized course and examination timetabling problems and have solved these datasets with our algorithms. We have also solved these datasets with genetic and tabu search algorithms and compared results of these four algorithms while using same termination criteria.

Rest of the chapter is organised in this way. We have presented experimental work on course timetabling problem in Section 7.1. In this section, first we have compared our memtic algorithm performance with other timetabling approaches on benchmark problem. In this section we have generated datasets for our generalized problem and have solved them with our proposed algorithms. These datasets are also solved with genetic and tabu search algorithms. We give detail comparison of the performance of these four algorithms. In Section 7.2, we have discussed examination timetabling problem and have compared results obtained by algorithms. In Section 7.3, we have conducted an analysis of the obtained results.

7.1 Course timetabling problem

In this section, we have talked about course timetabling problem. We have discussed a benchmark and have compared results of our memetic algorithm with other algorithms on this particular benchmark.

We have also generated datasets for generalized university course timetabling problems. We have solved these problems with our memetic and honey bee algorithms. We have also solved the same datasets with genetic and tabu search algorithms and finally have given a comparison of all these four algorithms with respect to each other.

7.1.1 Benchmark solved by proposed memetic algorithm

Curriculum based course timetabling problem is a track in International timetabling competition 2007 (ITC 2007) and this track is about weekly scheduling where university will publish curricula first. This model applies to University of Udine (Italy) and many other Italian universities. Datasets are taken from these universities and are modified little to make them general to use them as competition datasets. The problem consists of the following entities, number of days, number of periods per day, number of courses, number of teachers, number of rooms, capacity of each room, number of curricula and many other constraints.

The information about Udine course timetabling problem (ITC 2007: curriculum based course timetabling) [2] are given in Table 7.1. This table presents dataset number, dataset name, total number of rooms, total number of periods, total number of courses, the sum of their events in a week, the number of distinct enrolments (curricula).

Dataset	AKA	Rooms	Periods	Courses	Events	Curricula
comp01	Fis0506-1	6	30	30	160	14
comp02	Ing0203-2	16	25	82	283	70
comp03	Ing0304-1	16	25	72	251	68
comp04	Ing0405-3	18	25	79	286	57
comp05	Let0405-1	9	36	54	152	139
comp06	Ing0506-1	18	25	108	361	70
comp07	Ing0607-2	20	25	131	434	77
comp08	Ing0607-3	18	25	86	324	61
comp09	Ing0304-3	18	25	76	279	75
comp10	Ing0405-2	18	25	115	370	67

comp11	Fis0506-2	5	45	30	162	13
comp12	Let0506-2	11	36	88	218	150
comp13	Ing0506-3	19	25	82	308	66
comp14	Ing0708-1	17	25	85	275	60

Table 7.1: Showing dataset number, dataset name, total number of rooms, total number of periods, total number of courses, the sum of their events in a week and the number of curricula.

Some more characteristics about datasets are given in Table 7.2. They are as follows: frequency or the portion of period-room slots in use, utilisation in terms of used seats in percentage [32], numbers of edges in conflict graphs (CG), density in conflict graphs (CG) with vertices representing courses rather than events [33] and total unavailability constraints (sum of all unavailable periods for all courses, for example for 30 courses, if each course is unavailable for any 2 periods, then total unavailability constraints are 60)

Dataset	AKA	Frequency (used slots) %	Utilisation (used seats) %	Edges in CG (course - based)	Density of CG (course-based) %	Un availability constraints
comp01	Fis0506-1	88.89	45.98	53	12.18	53
comp02	Ing0203-2	70.75	46.28	401	12.07	513
comp03	Ing0304-1	62.75	38.30	342	13.38	382
comp04	Ing0405-3	63.56	33.22	212	6.88	396
comp05	Let0405-1	46.91	43.50	917	64.08	771
comp06	Ing0506-1	80.22	45.28	437	7.56	632
comp07	Ing0607-2	86.80	41.71	508	5.97	667
comp08	Ing0607-3	72.00	37.39	214	5.85	478
comp09	Ing0304-3	62.00	32.67	251	8.81	405
comp10	Ing0405-2	82.22	36.38	481	7.34	694
comp11	Fis0506-2	72.00	56.23	75	17.24	94
comp12	Let0506-2	55.05	35.06	1181	30.85	1368
comp13	Ing0506-3	64.84	38.14	216	6.50	468
comp14	Ing0708-1	64.71	34.78	368	10.31	486

Table 7.2: Frequency or the portion of period-room slots in use, utilisation in terms of used seats in percentage, numbers of edges in conflict graphs (CG), density in conflict graphs (CG) with vertices representing courses rather than events and total unavailability constraints.

Some more statistical quantities are presented in Table 7.3, it shows: average number of conflicts (Co), average teacher availability (TA), average number of lessons per curriculum per day (CL), average room occupation (RO). The feature Co (conflicts) counts the pairs of lessons that have clash and cannot be assigned at the same time (like same curriculum, same course and same teacher) divided by the total number of distinct pairs of lessons [19].

Dataset	AKA	CO	TA	CL	RO
comp01	Fis0506-1	13.2	93.1	3.24	88.9
comp02	Ing0203-2	7.97	76.9	2.62	70.8
comp03	Ing0304-1	8.17	78.4	2.36	62.8
comp04	Ing0405-3	5.42	81.9	2.05	63.6
comp05	Let0405-1	21.7	59.6	1.8	46.9
comp06	Ing0506-1	5.24	78.3	2.42	80.2
comp07	Ing0607-2	4.48	80.8	2.51	86.8
comp08	Ing0607-3	4.52	81.7	2	72
comp09	Ing0304-3	6.64	81	2.11	62
comp10	Ing0405-2	5.3	77.4	2.54	82.2
comp11	Fis0506-2	13.8	94.2	3.94	72
comp12	Let0506-2	13.9	57	1.74	55.1
comp13	Ing0506-3	5.16	79.6	2.01	64.8
comp14	Ing0708-1	6.87	75	2.34	64.7

Table 7.3: Aaverage number of conflicts (Co), average teacher availability (TA), average number of lessons per curriculum per day (CL) and average room occupation (RO)

We have compared the performance our algorithm with respect to other seven reference algorithms. These seven algorithms include ITC 2007 organizer’s algorithm which is

developed by de Cesco et al. [19], the algorithm of the winner of the competition Tomas Müller [35], the algorithm of Second position holder Zhipeng Lu and Jin-Kao Hao, the algorithm of third winner Mitsunori Atsuta et al. [41,42], the 4th place algorithm by Martin Josef Geiger [36], the 5th place algorithm of Michael Clark et al. [37] and the algorithm proposed by Abdullah and Turabieh[38].

Tomáš Müller used a Local search based algorithm using routines taken from the constraint solver library. He also used various neighborhood search algorithms to eliminate violations of hard and soft constraints.

Zhipeng Lü and Jin-Kao used iterated tabu search algorithm with kempe chains to solve competition datasets.

Mitsunori Atsuta, Koji Nonobe and Toshihide Ibaraki used constraint satisfaction problem solver incorporating tabu search and iterated local search.

Martin Josef Geiger used threshold accepting local search. Michael Clark, Martin Henz, Bruce Love used Repair-based local search.

The important thing to note about Abdullah and Turabieh algorithm is that this is the only algorithm among many others which used evolutionary algorithm to solve this competition problem. So this is our motivation to use evolutionary algorithm to solve these datasets. Thus we have constructed an evolutionary algorithm for our generalized model.

All the algorithms used the same stopping criteria (timeout condition 600 seconds), which was required by ITC-2007 competition [43,44]. Table 12 shows our Memetic algorithm's results in comparison with some other used techniques.

We have solved these competition datasets with memetic algorithm which uses genetic algorithm with local search. This algorithm is given in detail in the previous chapter. Our results are good when we compare it with contestant of the competition results. We also solved these datasets with genetic algorithm but results were not as good as these are with memetic algorithm. Maximum allowed time to solve these datasets was ten minutes. Some of the datasets are solved within this time limit but for some datasets it takes more time. We have fixed number of iterations to solve these datasets which are fifteen thousand for each dataset. In our proposed algorithm, we have used population size 100, mutation rate 7 while using 20 % killing with a local search which makes it slow but if we use steady state memetic algorithm (in which one offspring is generated in each generation), algorithm works quickly

but this decrease the performance of the algorithm. This means that in less time more iterations can be run but net objective function value would not be so good as with memetic algorithm. Our algorithm solves these datasets and produces good results but it takes more time to produce these results.

Dataset	Our MA	Müller [35]	Lü &Hao [41]	Atsuta [42]	Geiger [36]	Clark [37]	Cesco et al. [19]	Abdullah & Turabieh[38]
comp01	5	5	5	5	5	10	5	5
comp02	200	51	55	50	11	111	75	58
comp03	100	84	71	82	128	119	93	82
comp04	35	37	43	35	72	72	45	39
comp05	319	330	309	312	410	426	326	318
comp06	158	48	53	69	100	130	62	49
comp07	160	20	28	42	57	110	38	11
comp08	163	41	49	40	77	83	50	44
comp09	114	109	105	110	150	139	119	103
comp10	155	16	21	27	71	85	27	15
comp11	22	0	0	0	0	3	0	0
comp12	401	333	343	351	442	408	358	341
comp13	65	66	73	68	622	113	77	69
comp14	75	59	57	59	90	84	59	57

Table 7.4: Our memetic algorithm's (Our MA) results (minimize objective function value) in comparison with some other used techniques for International timetabling competition datasets (ITC-2007).

Our algorithm is originally made for the solution of generalized educational timetabling problem and is constructed when considering many hard and soft constraints. It is not constructed to focus on some specific problem. It has many type of parameters and sets in it which are making its performance bit slow for specific problem. But overall it gives promising performance and can solve large size datasets efficiently. We have made our memetic algorithm on the foundation of our genetic algorithm which we used to solve a

generic university course timetabling problem [25]. But our new generalized model has more constraints than our previously used problem for genetic algorithm.
Our algorithm is a two phase algorithm which satisfies the soft constraint in the first phase. Second phase is used to minimize the soft constraints while maintaining solution feasible. We hope that if we increase number of iterations of our algorithm, this will minimize the objective function value more.

Many authors had worked on this timetabling problem and produced very good solutions. These solutions can be seen from website [http://tabu.diegm.uniud.it/ctt/index.php]. Many ways were used to solve this curriculum based course timetabling problem including local search, tabu search, SAT based, Simulated annealing, mathematical programming, hybrid methods. If we look on the list of submitted results and proposed methods we will find only one author who used genetic algorithm to solve this problem [38]. Many authors constructed solutions for these datasets in last five years so some authors had good solutions on some datasets and others had good solutions on any other datasets but in our point of view overall good results were produced by Andrea Schaerf [http://tabu.diegm.uniud.it/ctt/index.php, [19]].

In this section we have generated data for our generalized problem and have solved these datasets with our proposed algorithms to compare their performance.

7.1.2 Description of the generalized course timetabling problem

In this section we have defined our generalized course timetabling problem having many constraints. Which constraint is used as hard and which constraint is used as soft is mentioned in below given tables (Table 7.5 and Table 7.6), if any constraint is used as a soft constraint then how much is the penalty value for its violation. We have solved this generalized timetabling problem to check the performance of our algorithms. Our generalized problem consists of two problems named Problem.1 and Problem.2.
Problem.1 is the generalized problem used in Ahmad et al. [25]. Problem.1 has same constraints as used in previous mentioned article but has different data. Here we have generated new data to check the performance of our proposed algorithms which is more difficult from previous data because of having more constraints and size of the problem datasets is also larger than previously mentioned data. Table 7.5 is showing constraints for

Problem.1.

Constraints	Hard	Soft
Every lesson should be scheduled	×	
Class must not be double booked	×	
Teacher must not be double booked	×	
Room must not be double booked	×	
Schedule course if room available	×	
Teacher unavailability	×	
Schedule on pre assigned periods (courses)	×	
No scheduling on forbidden periods	×	
Maximum working days for course		1
A class single lesson in a day		1
Class compactness		2
Max. daily lessons of a course		1
Course compactness		1
Assign course in adequate room type		1
Maximum number of lessons per day of a class		1
Same room used every time for a course		1
Minimum working days for course		5
Room capacity		Size of class - capacity of room ≥ 0

Table 7.5: Showing constraints for Problem.1 and provides details about every constraint whether it is used as hard or soft. If it is used as a soft constraint then it also tells about the amount of penalty used in case of violation.

We have used Problem.1 datasets to check the performance of our algorithms in detail. We have also solve Problem. 2 to show that how this problem with large number of constraints can be solved with these algorithms. But we perform our detailed experimental work to compare the performance of algorithms on Problem.1.

We have constructed Problem. 2 by adding some more constraints into Problem.1 and Problem.2 consists of all constraints of Table 7.5 and 7able 7.6. This is a more generalized

problem having many constraints.

Table 7.5 is showing constraints for Problem.1 and Table 7.6 is showing some more constraints. Problem 2 is the sum of Table 7.5 and Table 7.6 constraints; it means that Problem.2 has all constraints. Each integer under Soft in tables represent penalty for each violation of soft constraint and the symbol of $\times$ under Hard in table represent that constraint is used as hard constraint in problem.

Constraints	Hard	Soft
Free periods		1
Room compactness		1
Free rooms		1
Courses scheduled on same day		1
Courses scheduled on the different days		1
Precedence constraint among courses		1
Courses scheduled on the same period		1
Courses scheduled on different periods		1
Max. load for teacher per day		1
Maximum working days for teacher		1
Minimum working days for teacher		1

Table 7.6: Showing additional constraints used for Problem.2

7.1.3 Data generation for course timetabling problem

We have modelled a generalized educational timetabling problem. In this section we have generated data to check the performance of proposed methods for solution. For this purpose we have generated data where some sets and sub sets have fixed values and others have been chosen randomly. The purpose of this work is to make some datasets which have all these constraints and to solve them with proposed methods. Our proposed methods are two phase methods, so in first phase hard constraints are satisfied and these methods get feasible solution in first phase.

We have tried all constraints in preliminary experiments but excluded few of them in our experimental work. Because few constraints were contradicting each other so their use at a same time is not adequate and some constraints were making problem more complex. For example gap of specific periods in teacher courses, same class two courses should not be scheduled on consecutive periods and class can have maximum two consecutive courses can effect compactness constraints like room compactness, class compactness, course compactness. Some constraints like all the course of a class must be scheduled either in morning or afternoon session or course should not be scheduled in the last period of day was excluded to make search space little lenient because problem has already many constraints.

How we have generated our data for these problems, details of data generations are given as follows.
Total number of courses, classes, teachers, rooms, room types, days and number of periods per day are fixed. Each teacher has assigned at least one and at most two courses.

1. For each class, number of courses is randomly generated from 1 to 4; a course can be part of maximum 4 classes.
2. Number of students who attend a certain course is randomly chosen between 20 and 100.
3. For each course, we choose a random number 2 to 5 which shows the number of lessons of course.
4. For each course, it is required that it should not finish its lessons before a minimum number of days. We have taken randomly minimum number of day from 1 to 3.
5. Each room capacity is randomly generated from 30 to 120.
6. One lesson of each course is randomly pre assigned a period.
7. For each course, it is required that it should not be scheduled after a fix maximum number of days. We have taken randomly maximum number of day from 3 to 5.
8. The maximum daily number of teaching hours allowed for any class is 4.
9. For each course, maximum limit on number of teaching hours per day is randomly taken from 1 to 5.
10. Every course needs a desired type of room to occur. We have specified number of room types. We have allocated a room type to a course randomly.

11. For each room, we generate an availability matrix. In this matrix, we make each room has 5-10% unavailability.
12. For each teacher, we have certain number of unavailabilities. We have generated these unavailable periods randomly. These periods are 10% of the total number of periods.
13. Each teacher has maximum working days and minimum working days. We have assigned these values randomly from 2 to 4 days for maximum working days and from 1 to 2 for minimum working days.
14. On the day, on which a teacher has a course to teach, there is a maximum load of teacher for that day. The maximum working load is defined here in terms of working hours. For each teacher, we take it 5, so this is a trivial condition.
15. Each course has its 12 % randomly chosen unavailable periods, where its scheduling is forbidden.
16. For each period, we have defined the maximum number of classes occurred is taken equal to the number of rooms available at this period.
17. We have generated randomly 5, 2, 1 precedence relations (one precedence relation means that we have chosen two courses and this precedence relation is between every lesson of these two courses) respectively for large, medium, small data in such a way that the generated data should not be contradictory. For example if lesson *a* is preceded by *b* and *b* is preceded by a then these two precedence constraints are contradictory.

We can find similar precedence relationship in real world problems when we have course consisting of theory and practical work, practical work should occur after the theory lecture (each theory lesson follows practical work lesson). This relationship should be followed for every occurrence of theory and practical work.

18. Five percent lessons (from non conflicting courses) are grouped into two or three lessons subsets randomly which should scheduled together in same period. Each lesson can be part of at most one sub set.
19. Three percent lessons are selected randomly, grouped into two or three lessons subsets randomly which should not be scheduled together in same period. Each lesson can be part of at most one sub set.
20. One period is chosen randomly to be free.
21. One day is randomly chosen free for a teacher.

22. Five percent lessons are grouped into two or three lessons subsets randomly which should be scheduled together in same day. Each lesson can be part of at most one sub set.
23. Three percent lessons are selected randomly, grouped into two or three lessons subsets randomly which should not be scheduled together in same day. Each lesson can be part of at most one sub set.

Different constraints used in generalized problems are given in Tables 7.5 and 7.6. These tables provide details about constraint whether it is used as hard or soft. If it is used as a soft constrain then they also tell about the amount of penalty used in case of violation.

For these datasets, we have taken mutation rate 7, kill colony 20 percent, population size 100 because our preliminary experiments suggest that good results can be produced with these parameters. We allow 600 seconds for large problem datasets, 400 seconds for medium problem datasets and 100 seconds for small problem datasets.

We have generated certain data in terms of percentage. To obtain an integer value from its percentage, we use least integer function. For example if 5% of total lessons gives 6.4 lessons, so least integer function value of 6.4 be 7. The reason to do so is that in our problem values are in integers not in form of any real number. 6.4 lessons are unavailable for a period does not make any sense or teacher is unavailable for 5.2 periods. According to our data generation 5.2 unavailable periods mean 6 periods unavailability.

To check the performance and efficiency of the algorithms, we have firstly applied the algorithms on randomly generated datasets of Problem.1. We have generated 10 small, 10 medium and 10 large size problem datasets for this purpose. The program is coded in C and run on an Intel computer with 2.5 Ghz processor, 4GB RAM under windows operating system. We have run algorithm 5 times for each problem dataset to see its performance in detail.

7.1.4 Results of course timetabling Problem.1 datasets

These tables are showing results for small size, medium size and large size problem datasets. In each table C is number of course, T is number of teachers, K is number of classes, R is number of rooms, RT is number of room types, P is the total number of periods and D for

working days. After running each problem dataset 5 times, we have given minimum cost, maximum cost, average cost, S.D. (standard deviation), total number of lessons (events) and percentage of used slots and seats.

Small size datasets have C=20, T=16, K=6, R=4, RT=1, P=5, D=5. Medium size problem datasets have C=70, T=56, K=40, R=12, RT=3, P=5, D=5 and similarly large size problem datasets have C=125, T=100, K=70, R=20, RT=3, P=5, D=5.
In this section, we have given results obtained by honey bee mating, memetic, gentic and tabu search algorithms across 5 runs on each of the 10 datasets of small, medium, large size problem. We have taken mutation rate 7, kill colony 20 percent, population size 100, time limit for small datasets 100 seconds, time limit for medium datasets 400 seconds and time limit for large datasets is fixed 600 seconds.

7.1.4.1 Memetic algorithm

In this section, we have given results (Table 7.7, Table 7.8, Table 7.9) obtained by memetic algorithm. It is clear from results that small size problem is easy to solve in less amount of time than large problem. One can find near optimal solution with memetic algorithm in very short time.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	0	0	0	0	68	66	68
2	5	5	5	0	69	68	69
3	5	7	6	0.81	79	70	79
4	5	5	5	0	71	75	71
5	5	5	5	0	67	74	67
6	10	10	10	0	68	80	68
7	0	0	0	0	65	84	65
8	15	16	15.28	0.48	65	82	65
9	10	10	10	0	63	87	63
10	9	9	9	0	59	76	59

Table 7.7: Numerical results for small datasets obtained by memetic algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	46	53	49	3.16	225	83	75
2	26	35	31.75	4.27	254	81	84
3	30	32	31.25	0.95	229	76	76
4	45	53	48.75	3.30	243	83	81
5	38	51	43.25	6.02	248	82	82
6	38	46	41.75	3.30	243	75	81
7	35	49	44.5	6.45	242	77	80
8	47	53	50.25	3.20	231	72	77
9	35	39	36.5	1.73	233	69	77
10	56	62	59.5	2.51	250	81	83

Table 7.8: Numerical results for medium datasets obtained by memetic algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	154	169	158.16	5.49	411	77	82
2	144	162	153	6.85	426	78	85
3	128	135	131.4	2.88	423	73	84
4	121	126	123	2.16	421	78	84
5	133	165	145.5	14.05	433	75	86
6	145	164	154.5	9.39	412	76	82
7	251	264	259.5	6.85	457	82	91
8	169	183	178.25	6.39	463	75	92
9	184	215	200	14.07	467	85	93
10	130	163	144.25	13.72	409	80	81

Table 7.9: Numerical results for large datasets obtained by memetic algorithm

7.1.4.2 Honey bee mating algorithm

In this section, we have given results (Table 7.10, Table 7.11, Table 7.12) obtained by honey bee mating algorithm. We take number of queens 1, number of drones 99, number of mating flights 4, Size of queen's spermatheca 5, number of elite drones 4, number of common drones 16, Ne-mat 1, Nc-mat 4. Honey bee mating algorithm is also taking more time to solve large size problem than small and medium size problems.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	0	0	0	0	68	65	68
2	5	5	5	0	69	67	69
3	5	8	6.67	1.21	79	68	79
4	5	5	5	0	71	76	71
5	5	5	5	0	67	73	67
6	10	10	10	0	68	77	68
7	0	0	0	0	65	84	65
8	15	15	15	0	65	83	65
9	10	10	10	0	63	86	63
10	9	9	9	0	59	77	59

Table 7.10: Numerical results for small datasets obtained by honey bee mating algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	40	51	44.25	4.71	225	83	75
2	29	35	32.25	3.20	254	81	84
3	99	103	100.67	2.08	229	81	76
4	46	49	47	1.73	243	83	81
5	36	44	41	4.35	248	82	82
6	36	40	38	2.0	243	76	81

7	40	48	44.33	4.04	242	78	80
8	51	54	51.67	2.08	231	73	77
9	36	40	38.67	2.30	233	69	77
10	55	57	56	1.0	250	81	83

Table 7.11: Numerical results for medium datasets obtained by honey bee mating algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	150	158	153.25	3.94	411	77	82
2	145	153	148.25	3.59	426	79	85
3	123	125	123.75	0.96	423	73	84
4	112	123	117.0	5.56	421	78	84
5	129	142	135.0	6.55	433	74	86
6	134	146	138.67	6.42	412	77	82
7	205	232	216.33	14.01	457	81	91
8	167	188	176.0	10.81	463	74	92
9	174	197	183.0	12.28	467	85	93
10	120	138	127.66	9.29	409	81	81

Table 7.12: Numerical results for large datasets obtained by honey bee mating algorithm

7.1.4.3 Genetic algorithm

In this section, we have given results (Table 7.13, Table 7.14, Table 7.15) obtained by genetic algorithm. Percentage of used slots is same for all results of any dataset but percentage of used seats can be different because used seats depend upon assignment of room and it can be different in different timetable of same dataset.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	1	4	1.71	1.25	68	67	68
2	5	7	5.5	1.00	69	69	69
3	11	13	11.8	0.84	79	68	79
4	5	7	6.5	0.84	71	76	71
5	5	8	6	1.09	67	73	67
6	10	13	10.75	1.50	68	77	68
7	1	4	1.5	1.22	65	83	65
8	15	16	15.25	0.50	65	83	65
9	10	10	10	0	63	88	63
10	9	9	9	0	59	78	59

Table 7.13: Numerical results for small datasets obtained by genetic algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	427	460	440.25	14.08	225	70	75
2	195	226	209.25	13.96	254	82	84
3	110	116	113.75	2.63	229	82	76
4	54	69	63.2	7.53	243	84	81
5	43	66	52.4	8.96	248	82	82
6	41	52	46.67	5.50	243	78	81
7	47	50	48.25	1.50	242	80	80
8	49	58	53.50	5.19	231	73	77
9	35	46	39.5	4.79	233	73	77
10	63	77	70.75	6.44	250	81	83

Table 7.14: Numerical results for medium datasets obtained by genetic algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	288	322	300.5	15.60	411	78	82
2	235	268	257.5	15.28	426	77	85
3	180	205	193.33	12.58	423	74	84
4	252	274	263.0	8.98	421	79	84
5	185	207	198.0	9.30	433	76	86
6	188	205	196.0	8.54	412	78	82
7	437	461	466.33	12.85	457	81	91
8	317	326	320.67	4.72	463	74	92
9	369	377	372.0	4.35	467	85	93
10	226	242	236.0	8.71	409	79	81

Table 7.15: Numerical results for large datasets obtained by genetic algorithm

7.1.4.4 Tabu search algorithm

In this section, we have given results (Table 7.16, Table 7.17, Table 7.18) obtained by tabu search algorithm. We take tabu list length 5. Standard deviation for small size problem is less and as problem size increases it also increases.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	1	1	1.0	0	68	67	68
2	5	5	5.0	0	69	67	69
3	11	14	13.0	1.73	79	67	79
4	6	7	6.33	0.58	71	76	71
5	6	6	6.0	0	67	73	67
6	10	10	10.0	0	68	80	68
7	0	0	0	0	65	85	65
8	15	16	15.67	0.58	65	85	65
9	10	10	10	0	63	87	63
10	9	9	9	0	59	77	59

Table 7.16: Numerical results for small datasets obtained by tabu search algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	139	141	139.67	1.15	225	82	75
2	295	355	314.5	27.63	254	79	84
3	209	252	229.33	21.59	229	82	76
4	149	220	181.33	35.92	243	82	81
5	160	215	187.0	22.46	248	83	82
6	119	157	138.25	18.13	243	76	81
7	112	133	121.50	8.66	242	77	80
8	80	83	81.67	1.50	231	73	77
9	79	80	79.66	0.58	233	71	77
10	310	358	335.33	24.11	250	81	83

Table 7.17: Numerical results for medium datasets obtained by tabu search algorithm

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	437	493	475.50	25.92	411	77	82
2	358	404	379.33	23.18	426	78	85
3	239	266	249.67	14.36	423	73	84
4	377	407	393.0	15.10	421	78	84
5	328	393	364.33	33.17	433	74	86
6	248	282	262.33	17.61	412	76	82
7	946	1092	1002.0	67.97	457	80	91
8	659	724	685.2	29.96	463	74	92
9	957	1143	1036.75	88.92	467	83	93
10	319	329	325.25	5.29	409	81	81

Table 7.18: Numerical results for large datasets obtained by tabu search algorithm

7.1.4.5 Minimum cost Comparison of all algorithms

In this section, we have given comparison of minimum cost results obtained by all algorithms (Table 7.19, Table 7.20, Table 7.21). Results of Honey bee mating and memetic algorithms are same for small datasets, while results of genetic algorithms are better than tabu search algorithm.

For medium datasets, on some datasets honey bee mating algorithm is giving good results but for others memetic algorithm's performance is better. Memtic algorithm has given good results on 7 datasets while honey bee mating algorithm has given good results on 3 datasets. Thus for medium datasets, memetic algorithm has performed better than honey bee mating algorithm. Genetic algorithm performance is better than tabu search algorithm but for one dataset genetic algorithm is exceptionally performing well.

For large datasets, honey bee mating algorithm's performance is far better than memtic algorithm. Honey bee mating algorithm results are better than memetic algorithm for 9 datasets out of total 10.

We think honey bee mating algorithm's performance is better due to its ability to explore and exploit the search space simultaneously by using probabilistically guided search by the queen's transition and by employing local search at each iteration. The queen (current fittest solution) is the best individual and every new brood is the composition of some parts of the drone's genotype with parts of the queen genotypes. So it is hoped that it will evolve superior solutions. Again genetic algorithm is performing better than tabu search algorithm.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	0	5	5	5	5	10	0	15	10	9
M.A	0	5	5	5	5	10	0	15	10	9
G.A	1	5	11	5	5	10	1	15	10	9
T.S.A	1	5	11	6	6	10	0	15	10	9

Table 7.19: Minimum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for small datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	40	29	99	46	36	36	40	51	36	55
M.A	46	26	30	45	38	38	35	47	35	36
G.A	427	195	110	54	43	41	47	49	35	63
T.S.A	139	295	209	149	160	119	112	80	79	310

Table 7.20: Minimum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for medium datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	150	145	123	112	129	134	205	167	174	120
M.A	154	144	128	121	133	145	251	169	184	130
G.A	288	235	180	252	185	188	437	317	369	226
T.S.A	437	358	239	377	328	248	946	659	957	319

Table 7.21: Minimum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for large datasets.

7.1.4.6 Maximum cost Comparison of all algorithms

In this section, we have given comparison of maximum cost results obtained by all algorithms (Table 7.22, Table 7.23, Table 7.24). For small datasets, results of honey bee mating and memtic algorithms are same. Genetic algorithm has given more maximum cost values than tabu search algorithm. For medium datasets, honey bee mating and memtic algorithms results are almost same. Tabu search algorithm is giving more maximum cost values than genetic algorithm. This is opposite to small size instances where genetic algorithm costs are more.

In our opinion, for complex, large size problem and over big span of time genetic algorithm's operators and its nature inspired behaviour giving it edge on local search based algorithm. For large datasets honey bee mating algorithm maximum cost is less than memetic algorithm for 9 datasets. Thus honey bee has produced better results than memetic algorithm for large instances in terms of both maxim and minimum cost comparison.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	0	5	8	5	5	10	0	15	10	9
M.A	0	5	7	5	5	10	0	16	10	9
G.A	4	7	13	7	8	13	4	16	10	9
T.S.A	1	5	14	7	6	10	0	16	10	9

Table 7.22: Maximum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for small datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	51	35	103	49	44	40	48	54	40	57
M.A	53	35	32	53	51	46	49	53	39	62
G.A	460	226	116	69	66	52	50	58	46	77
T.S.A	141	355	252	220	215	157	133	83	80	358

Table 7.23: Maximum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for medium datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	158	153	125	123	142	146	232	188	197	138
M.A	169	162	135	126	165	164	264	183	215	163
G.A	322	268	205	274	207	205	461	326	377	242
T.S.A	493	404	266	407	393	282	1092	724	1143	329

Table 7.24: Maximum cost comparison of honey bee mating, memetic, genetic and tabu search algorithms for large datasets.

7.1.4.7 Average cost Comparison of all algorithms

In this section, we have given ccomparison of average cost results obtained by all algorithms (Table 7.25, Table 7.26, Table 7.27). Average cost for honey bee mating and memetic algorithms are almost same, average cost for tabu search algorithm is better than genetic algorithm for small dataset. For medium datasets, honey bee mating algorithm average cost is better than memetic algorithm for 6 datasets.

Average of genetic algorithm is better than tabu search algorithm. We have seen from our results that tabu search algorithm perform well for small datasets but its performance decreases as size of dataset increases comparing with genetic algorithm. For large datasets average cost of honey bee mating algorithm if far better than memetic algorithm and similarly genetic algorithm is far better than genetic algorithm for large size datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	0	5	6.6	5	5	10	0	15	10	9
M.A	0	5	6	5	5	10	0	15.2	10	9
G.A	1.7	5.5	11.8	6.5	6	10.7	1.5	15.2	10	9
T.S.A	1	5	13	6.3	6	10	0	15.6	10	9

Table 7.25: Average cost comparison results of honey bee mating, memetic, genetic and tabu search algorithms for small datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	44.	32.2	100.6	47	41	38	44.3	51.6	38.6	56
M.A	49	31.7	31.2	48.7	43.2	41.7	44.5	50.2	36.5	59.5
G.A	440.2	209.2	113.7	63.2	52.4	46.6	48.2	53.5	39.5	70.7
T.S.A	139.6	314.5	229.3	181.3	187	138.2	121.5	81.6	79.6	335.3

Table 7.26: Average cost comparison results of honey bee mating, memetic, genetic and tabu search algorithms for medium datasets.

Datasets	1	2	3	4	5	6	7	8	9	10
H.B.M.A	153.2	148.2	123.7	117	137	138.6	216.3	176	183	127
M.A	158.1	153	131.4	123	145.5	154.5	259.5	178.2	200	144.2
G.A	300.5	257.5	193.3	263	198	196	466.3	320.6	372	236
T.S.A	475.5	379.3	249.6	393	364.3	262.3	1002	685.2	1036.7	325.2

Table 7.27: Average cost comparison results of honey bee mating, memetic, genetic and tabu search algorithms for large datasets.

7.1.5 Results of course timetabling Problem.2 datasets

In this section, we have given results obtained by honey bee mating and memetic algorithms (Table 7.28, Table 7.29) across 5 runs on each of the 10 large size datasets of Problem.2. We have taken mutation rate 7, kill colony 20 percent, population size 100 and time limit for large datasets is fixed 600 seconds.

We have taken number of queens 1, number of drones 99, number of mating flights 4, Size of queen's spermatheca 5, number of elite drones 4, number of common drones 16, Ne-mat 1, Nc-mat 4 for honey bee mating algorithm. These all parameters and their values are same as are taken in Problem.1 in previous section. Datasets are also same but we have added some more constraints in these datasets.

We have not generated small and medium size instances for this problem because we want to see a glimpse of results for only large datasets as all size datasets have been discussed thoroughly in previous section results. Our purpose for these results is that to see the behavior of our proposed algorithms for more generalized complex problem that is why, we have solved Problem.2 only with our memetic algorithm and honey bee algorithms for large

datasets only.

7.1.5.1 Memetic algorithm

We were expecting that increase in constraints in problem will increase the value of objective function. Now values of objective function have been increased than objective function values of Problem.1 large size datasets. For large size problems, memetic algorithm has produced better results for five datasets than honey bee mating algorithm, while honey bee has constructed better results for four datasets than memetic algorithm.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	561	627	599	29.2	463	59	92
2	423	489	445.4	27.1	407	66	81
3	468	587	533.8	45.8	468	64	93
4	559	617	596.2	23.7	452	73	90
5	442	448	446	2.3	419	70	83
6	470	505	487	15.7	441	64	88
7	447	463	457	6.1	420	70	84
8	654	703	680.2	20.9	428	70	85
9	696	791	726.6	38.9	440	66	88
10	683	776	734.4	36.4	447	69	89

Table 7.28: Numerical results for large datasets obtained by memetic algorithm

7.1.5.2 Honey bee mating algorithm

When one increases number of constraints in a dataset its difficulty level increases. These large size datasets are really good to analyze the performance of algorithms. In Problem.1, honey bee mating algorithm performance was better for 9 datasets but by adding more constraints in problem, now honey bee algorithm has produced better results for four datasets. Similarly, there is a change in results for maximum cost of algorithms. Honey bee mating algorithm was better for 9 datasets in Problem.1 but now memetic algorithm has constructed better solutions for 9 datasets. However, performance of both algorithms is good.

Problem dataset	Minimum cost	Maximum cost	Average cost	S.D.	No. of events	% of used seats	% of used slots
1	539	647	591.4	42	463	60	92
2	423	504	446.4	32.7	407	66	81
3	505	592	545.6	32.5	468	64	93
4	560	660	583.8	38.2	452	72	90
5	446	508	471	23	419	69	83
6	492	516	500.2	9.5	441	64	88
7	462	487	480	11.9	420	69	84
8	646	733	674.8	35.9	428	70	85
9	695	709	700	5.7	440	66	88
10	679	787	714.4	41.9	447	69	89

Table 7.29: Numerical results for large datasets obtained by honey bee mating algorithm

7.2 Examination timetabling problem

We have also solved examination timetabling problem with our proposed algorithms. In this section, we have discussed our experimental work on this problem.

7.2.1 Data generation for Examination timetabling problem

In this section we have generated data for examination timetabling and have checked performance of proposed methods. For this purpose we have generated data where some sets and sub sets have fixed values and others have been chosen randomly. Examination timetabling problem is different from course timetabling so some sets and data pattern will be different from course timetabling problem.

In our mathematical model we have added invigilator and have added the constraints related to invigilator but in our experimental work we do not use them. One can add them to in the implementation phase on the same lines as we have added them for course timetabling problem but we have left them at the moment. The other reason is that in literature normally examination problem was dealt without invigilator inclusion. Similarly in this experimental

work we do not take different types of rooms as we take for course timetabling but these can also be added easily on the same lines like course timetabling problem

Details of data generations are given as follows.
Total number of examinations, classes, rooms, days, number of periods per day is fixed.
1. For each class, number of examinations is randomly generated from 2 to 5; an examination can be part of maximum 25 different classes.
2. Number of students who take examination is randomly chosen between 5 and 100.
3. Each room capacity is randomly generated from 20 to 100.
4. Duration of each examination is randomly chosen from 1 to 4.
5. Duration of each period is also randomly generated from 1 to 4.

For large data

Here we present some more details about our data which are different for large, medium and small problems.
1. Number of examinations for large problem are 125.
2. Number of classes for large problem is taken 70, these classes are generated randomly (each class has 2 to 5 examinations chosen randomly).
3. Total number of rooms is 9.
4. Number of days is 4.
5. Number of periods is also 4.
6. Number of examination which should be the sole occupier of the room is 12 (randomly chosen).
7. We have selected randomly 70 pairs of examinations which have precedence relation to satisfy.
8. We have chosen 35 pairs of examinations which have coincidence relation to satisfy.
9. 3 rooms are unavailable (chosen randomly).
10. 2 periods are unavailable (chosen randomly).
11. Maximum examinations of a class per day limit is set 3.
12. Maximum examinations per room limit is set 3.
13. Allowed time for large problem is 400 seconds.

For medium data

1. Number of examinations for medium problem are 70.
2. Number of classes for medium problem is taken 40, these classes are generated randomly (each class has 2 to 5 examinations chosen randomly).
3. Total number of rooms is 5.
4. Number of days is 4.
5. Number of periods is also 4.
6. Number of examination which should be the sole occupier of the room is 7 (randomly chosen).
7. We have selected randomly 40 pairs of examinations which have precedence relation to satisfy.
8. We have chosen 20 pairs of examinations which have coincidence relation to satisfy.
9. Two rooms are unavailable (chosen randomly).
10. One period is unavailable (chosen randomly).
11. Maximum examinations of a class per day limit is set 3.
12. Maximum examinations per room limit is set 2.
13. Allowed time for medium problem is 200 seconds.

For small data

1. Number of examinations for small problem are 20.
2. Number of clases for small problem is taken 6, these classes are generated randomly (each class has 2 to 5 examinations chosen randomly).
3. Total number of rooms is 2.
4. Number of days is 4.
5. Number of periods is also 4.
6. Number of examination which should be the sole occupier of the roon is 2 (randomly chosen).
7. We have selected randomly 9 pairs of examinations which have precedence relation to satisfy.
8. We have chosen 4 pairs of examinations which have coincidence relation to satisfy.
9. One room is unavailable (chosen randomly).
10. One periods is unavailable (chosen randomly).

11. We do not put limit of maximum examinations of a class per day limit for small problem datasets because maximum available rooms for these problems are 2.
12. Maximum examinations per room limit is set 2.
13. Allowed time for small problem is 100 seconds.

Constraint	**Hard**	**Soft**
Every exam schedule once	×	
Every exam schedule in a unique room	×	
Every exam schedule in a unique period	×	
Max. one exam of a class in a period	×	
Duration of exam $\leq$ to the duration of period	×	
Precedence constraint		1
Exams exclusion		1
Exams coincidence		1
Exam sole occupier of room		1
Two Exams in a Row		1
Two Exams in a day (on non consecutive periods)		1
Big exams scheduled first		1
Spare rooms		1
Spare periods		1
No mixed duration		1
Max. exams of a class per day		1
Max. exams in a period		1
Room capacity		Size of class - capacity of room ≥ 0

Table 7.30: Constraints for examination timetabling problem

For examination mutation rate 3, population size 100, 50 percent and time 400, 200, 100 seconds for large, medium and small respectively.
Many other constraints like pre assigned periods to examinations, examination on the same day, examination on the different days, specify any room for any examination, specified spread of examination, room types etc. can be added and solved easily in a similar way as these are dealt in course timetabling problem.

7.2.2 Results of examination timetabling problem datasets

In this section we have presented the results obtained from our algorithms by using above mentioned generated data. For large examination timetabling datasets, number of generated examinations is 125 and total number of used slots for this data is 86%, for medium date number of examinations is 70 and 87% slots are used and for small datasets number of examinations are 20 and percentage of used slots for these problems is 62%. The remaining parameters are the same as used for course timetabling problem.

7.2.2.1 Minimum cost comparison of all algorithms

Minimum cost results for small datasets have shown almost same performance for all algorithms. For medium datasets, honey bee mating algorithm has shown good performance than memetic algorithm for 3 datasets and memetic algorithm's performance is better for one dataset than honey bee mating algorithm but on remaining datasets their performance is same. For large datasets, honey bee algorithm performance is better for 5 datasets than memetic algorithm while memtic algorithm is showing better performance on 4 datasets than honey bee mating algorithm. Genetic algorithm performance is lower than honey bee mating and memetic algorithms.

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	1	5	2	4	1	0	4	4	0	0
M.A	1	5	2	4	1	0	4	4	0	0
G.A	2	5	2	4	1	0	4	4	3	0

Table 7.31: Minimum cost comparison of all algorithms for small datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	48	47	117	47	54	47	66	131	46	74
M.A	48	46	117	47	54	47	68	131	63	75
G.A	61	51	132	49	59	50	79	159	65	79

Table 7.32: Minimum cost comparison of all algorithms for medium datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	162	156	37	155	140	236	80	53	152	30
M.A	126	156	33	154	160	255	82	63	146	42
G.A	737	1040	603	719	429	1456	625	358	683	404

Table 7.33: Minimum cost comparison of all algorithms for large datasets

7.2.2.2 Maximum cost comparison of all algorithms

Maximum cost is same for honey bee mating and memetic algorithms for small datasets. For medium datasets, two datasets solved by honey bee mating algorithm have given more cost than memetic algorithm while memetic algorithm has shown more cost on one dataset.
Memetic algorithm has produced more cost for 8 datasets for large size datasets. Honey bee mating algorithm performance is better than memetic algorithm in terms of maximum cost comparison for large datasets. Genetic algorithm's overall performance is lower than honey bee mating and memetic algorithms for all datasets of all sizes.

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	1	5	2	4	1	0	4	4	0	0
M.A	1	5	2	4	1	0	4	4	0	0
G.A	2	5	2	4	1	0	4	4	3	0

Table 7.34: Maximum cost comparison of all algorithms for small datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	49	57	118	48	55	48	69	141	49	75
M.A	49	57	118	48	55	48	68	140	65	75
G.A	93	72	140	54	77	52	99	179	80	90

Table 7.35: Maximum cost comparison of all algorithms for medium datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	280	176	48	165	168	264	86	78	157	49
M.A	269	190	69	168	170	291	117	83	176	49
G.A	1125	1328	919	999	534	1595	832	475	916	487

Table 7.36: Maximum cost comparison of all algorithms for large datasets

7.2.2.3 Average cost comparison of all algorithms

Average cost of honey bee mating algorithm and memetic algorithm is same for small datasets while average cost of genetic algorithm is more than other two algorithms. Honey bee mating algorithm average cost is better on 6 datasets and memtic algorithm is performed better on 4 datasets for medium datasets. Honey bee mating algorithm has performed better on 8 datasets than memetic algorithm and memetic algorithm is performed better on 2 datasets.

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	1	5	2	4	1	0	4	4	0	0
M.A	1	5	2	4	1	0	4	4	0	0
G.A	2	5	2	4	1	0	3	4	3	0

Table 7.37: Average cost comparison of all algorithms for small datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	48.6	50	117.2	47.5	54.2	47.4	67	135.8	48	74.7
M.A	48.3	49.6	117.3	47.3	54.6	47.6	68	134.6	64	75
G.A	75.3	58.6	135.3	51	68.6	51.3	86	170	70.2	85.6

Table 7.38: Average cost comparison of all algorithms for medium datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	204.5	164.0	42.7	157	155.6	254.3	82.6	63.6	154.3	40.1
M.A	196.0	169.6	46.6	148	165.3	275.2	101.2	74.7	158.2	44.6
G.A	935	1142	782	850	476.6	1509.3	726.3	426.6	795.6	453.3

Table 7.39: Average cost comparison of all algorithms for large datasets

7.2.2.4 Standard deviation comparison of all algorithms

For small datasets, standard deviation is 0 for all algorithms. For medium datasets, honey bee mating algorithm is showing better results for 7 datasets while memetic algorithm is better on 3 datasets. Honey bee mating standard deviation is less for 5 datasets and on remaining 5 datasets memetic algorithm is showing less deviation value.

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	0	0	0	0	0	0	0	0	0	0
M.A	0	0	0	0	0	0	0	0	0	0
G.A	0	0	0	0	0	0	0	0	0	0

Table 7.40: Standard deviation comparison of all algorithms for small datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	0.52	4.34	0.50	0.55	0.45	0.55	1.22	5.21	1.1	0.5
M.A	0.58	6.35	0.58	0.58	0.58	0.58	0.0	4.73	1	0
G.A	16.26	11.59	4.16	2.64	9.07	1.15	11.27	10.15	5.93	5.8

Table 7.41: Standard deviation comparison of all algorithms for medium datasets

	1	2	3	4	5	6	7	8	9	10
H.B.M.A	49.22	9.09	5.56	5.0	14.2	15.8	3.06	12.8	2.52	8.13
M.A	45.69	13.54	13.79	8.08	5.03	15.19	14.52	9.54	14.84	3.79
G.A	194.1	161.07	162.26	140.87	53.16	74.93	103.56	61.09	116.69	43.69

Table 7.42: Standard deviation comparison of all algorithms for large datasets

7.3 Discussion on Results and conclusion

We have used certain time limits on our algorithms during testing of our datasets and we consider this is fair for these sizes of problems. These are 100, 400 and 600 seconds of CPU time for the small, medium and large datasets respectively. We have used hardware Pentium

IV 2.5Ghz processor with 4GB RAM under a Windows operating system for these experiments (and indeed for all experiments described in this thesis).

The behavior of these algorithms with the small, medium and large datasets in our experiments is actually noticed different in each case. Algorithms performance is well across the set of small datasets and there is less difference of performance for these datasets. Our algorithms are two-stage algorithms, which operates by first constructing a population of fully feasible timetables, and then evolves these whilst always remaining in feasible areas of the search space.

Our initial motivations for designing an evolutionary algorithm were as follows:
(1) Even though many different types of metaheuristic algorithm were submitted to the International Timetabling Competition, interestingly none of the entrants chose to make use of any sort of evolutionary technique.
(2) The other purpose of this algorithm was to get a feasible timetable in relatively small amounts of time by using two phase procedure.

We therefore consider it interesting to develop an evolutionary algorithm to solve our generalized timetabling problem that can follow this two phase approach and we can check its performance.

Our preliminary tests show that existing bench mark datasets (of ITC 2007: curriculum based course timetabling problem) can be easily solved with this algorithm for obtaining feasible solutions (solution that satisfies the hard constraints only). We have generated some datasets for this purpose. We have observed that if we increase the killing rate it would decrease the speed of algorithm but it will decrease objective function value in less number of iterations.

Our population based algorithms are two phase algorithms. They eliminate hard constraints in first phase and try to satisfy soft constraints in second phase. The first phases take only 3-5 % of total time used. While the second phase takes the rest of the time.

Cost on y-axis and number of iterations on x-axis

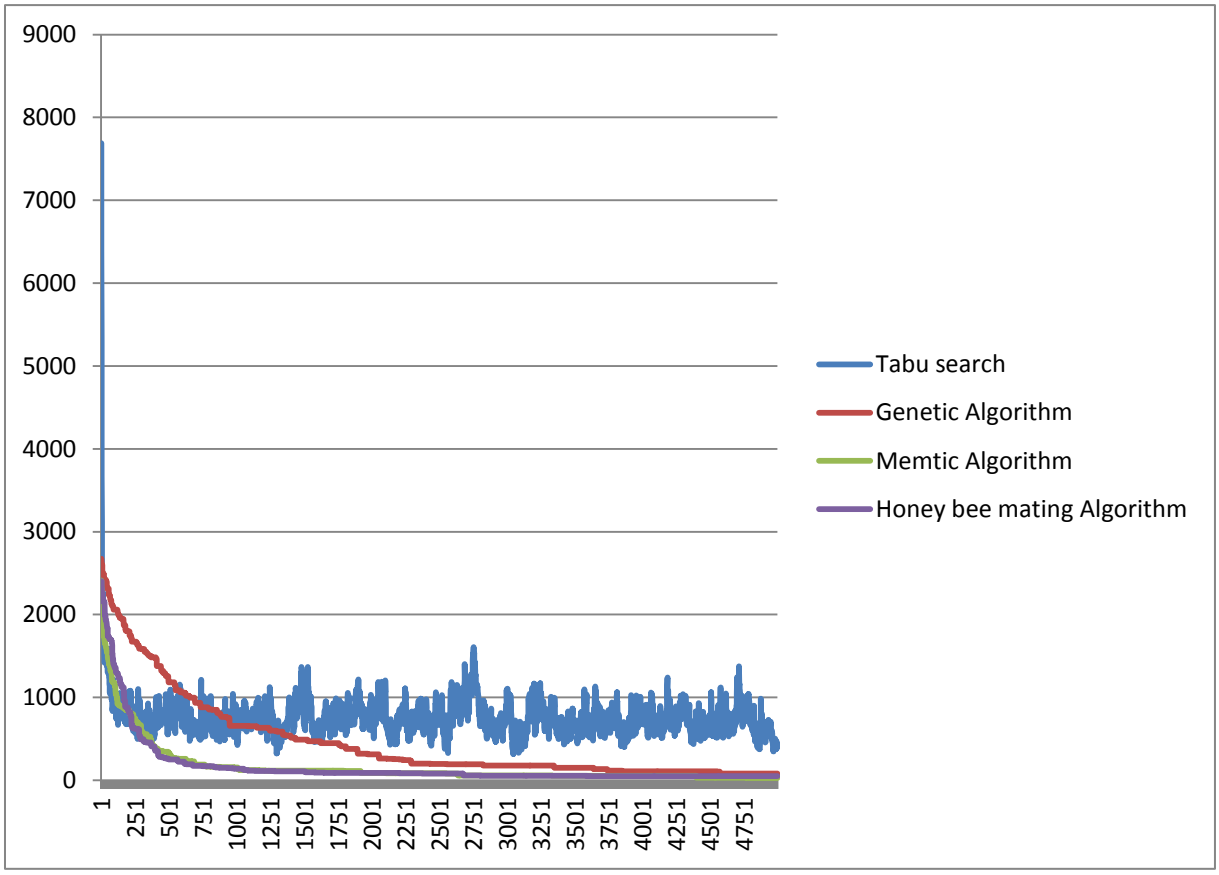


Figure 7.1: is showing the behaviour of the algorithms for medium sized dataset. These algorithms are run for 5000 iterations. X-axis represents number of iterations and Y-axis represents the cost at each iteration. The mutation rate is set to 7, kill colony 20 percent and population size 100.

The local search based algorithm (Tabu search) performs in single phase. It try to satisfy hard and soft constraints simultaneously. In tabu search, we have taken length of tabu list equal to 5. Figure 7.1 shows that tabu search is performing worst. The possible reason can be that in this tabu search, we have not used aspiration criteria or any guided search strategy for neighbourhood moves. By doing so, the search process may loos some good solutions.

In our preliminary experiments, we found that solutions obtained with simple local search in start are better than genetic algorithm but after certain time local search may trap in local minima and genetic algorithm continues its improvement as number of iterations increases. We think it is due to large number of population and the special genetic operators; these genetic operators provide diversity to genetic algorithm and also help to avoid from local minima.

Cost on y-axis and number of iterations on x-axis

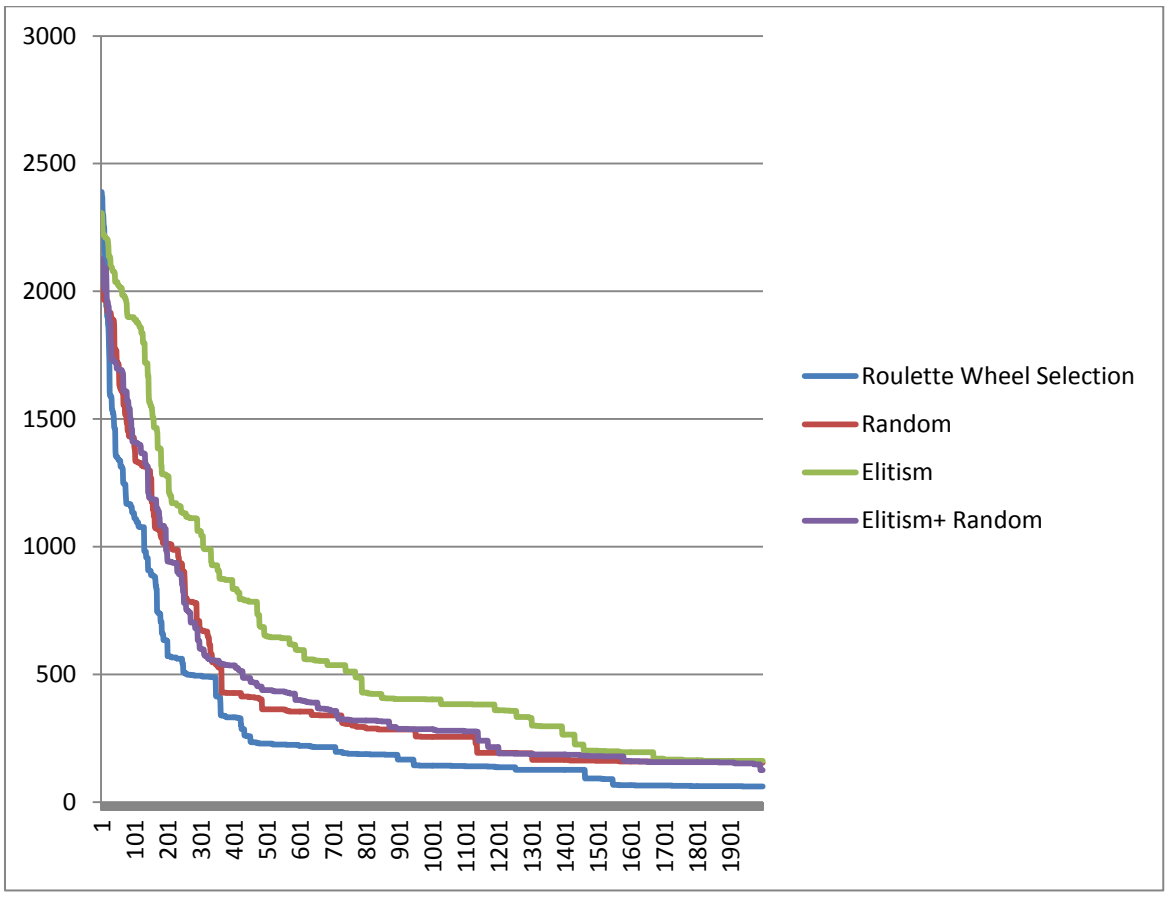


Figure 7.2: Showing the comparison of different selection operators for memetic algorithm

The figure (Figure 7.1) shows that memetic algorithm and honey bee mating algorithms are performing well. But Honey bee mating algorithm performance is better towards the end of iterations. We think that it is due to fixing one solution as queen for mating in the whole generation. This can reduce the diversity of solutions as compared to the memetic algorithm where we choose two different parents for each generation. We have run these algorithms once on one medium sized dataset. As in literature, normally ten to twenty runs are performed on one dataset. Then they get results for average cost of all the runs. But we have run these algorithms in our preliminary experiments during their construction and also for our small, medium and large datasets. Detail performance of these algorithms can be found from the results. We have drawn a graph of a medium sized dataset to show the general behaviour of these algorithms.

In figure 7.2, we have shown the comparison of different selection operators while using memetic algorithm. From this figure one can notice that roulette wheel selection is performing best for this algorithm. We think the reason of this best performance is that the roulette wheel selection provides more chance of survive to fittest individuals than weaker ones. Thus these fitter individuals have a better probability of survival and go forward for the next generation through the mating pool. At the same time weaker individuals also have chances of selection which also may be useful for future generations. Selection operator which uses elitism criteria does not perform well at start but at end its performance becomes bit acceptable. Similarly random and elitism plus random operators work well for algorithm but at the end the performance of elitism plus random selection operator performs well than random operator.

Cost on y-axis and time on x-axis

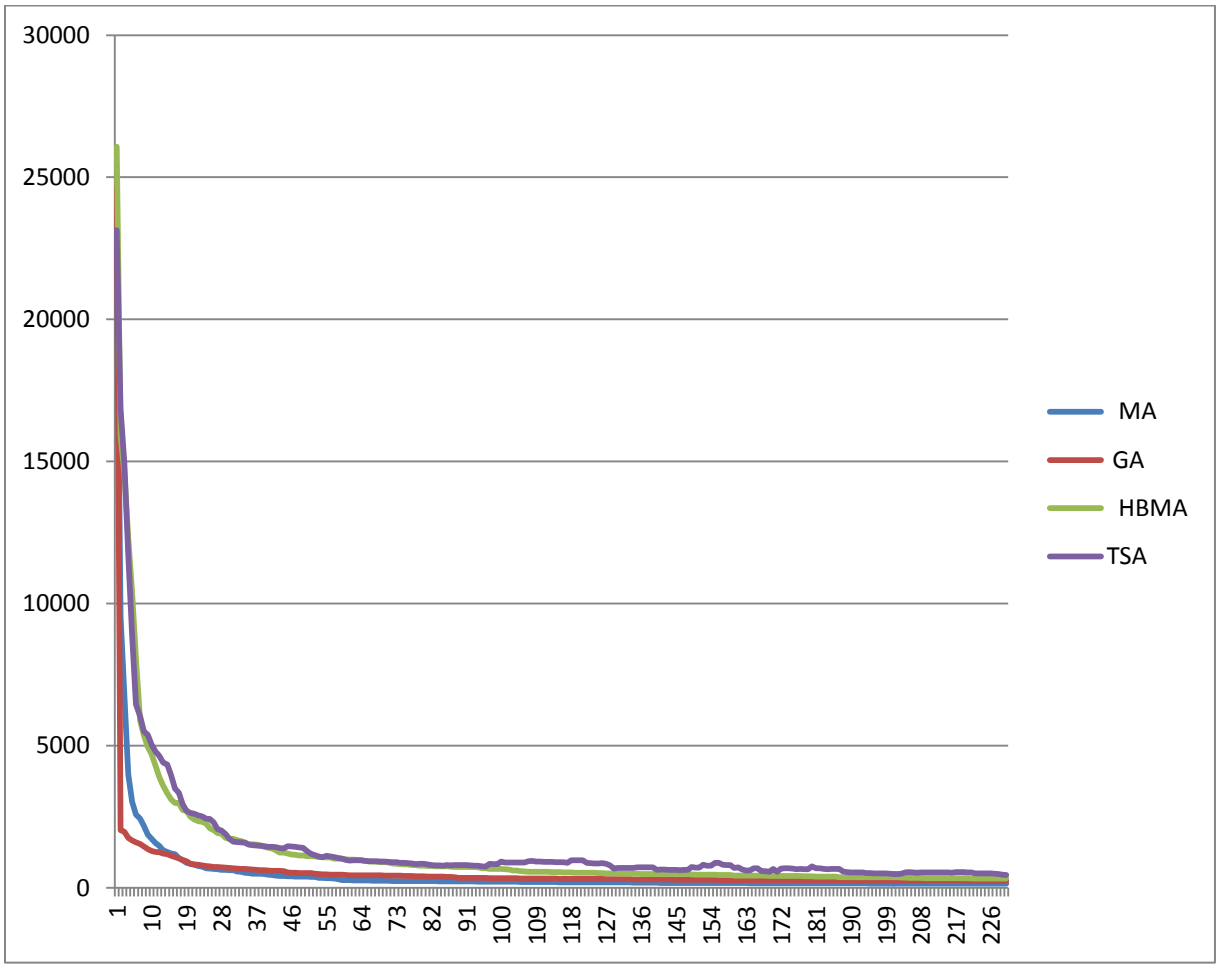


Figure 7.3: Time and cost comparison of all algorithms

In figure 7.3 cost-time comparison of these algorithms can be seen. Tabu search algorithm starts with big amount of objective value and in starting few seconds it reduces this cost value

quickly but then it reduces the value slowly. Genetic algorithm starts with relatively small cost value but after few seconds memetic algorithm and honey bee mating algorithms reach genetic algorithm and perform better till end.

Our experiments on datasets have shown that tabu search and genetic algorithm performance is good on small datasets, it constructs good solutions in comparison with memetic and honey bee algorithms. But large datasets affect their performance and tabu search algorithm could not produce good results on these problems. Although genetic algorithm performs relatively well than tabu sarch algorithm. We have found best results with honey bee mating and memetic algorithms.

GENERAL CONCLUSION

In this thesis we have examined various algorithms for university and examination timetabling problems. We have given a general conclusion of thesis and future perspectives here.

In Chapter 1, we have given an introduction of educational timetabling problems and the approaches used to solve these problems. We have discussed different types of educational timetabling problems with different types of constraints which makes this type of problem really difficult. We have focused on different metaheuristics used in literature to solve timetabling problems like local search based and population based metaheuristics because we have proposed metaheuristics to solve timetabling problems.In Chapter 2, We have formulated two mathematical formulations of course timetabling problem which are the prototype of single-mode RCPSP, which are linear integer programming models and could be solved by using linear programming solvers.

We have given a generalized mathematical formulation for university course timetabling problem in Chapter 3. We have written many constraints of different university environments in their mathematical relations. For that purpose, we have defined sets, subsets, different parameters and decision variables needed for formulating this generic model. These constraints are classified in six sets namely, hard constraints, period related constraints, room related constraints, class related constraints, course related constraints and teacher related constraints. We have also discussed how different objective functions could be written from soft constraint relations by using this generic model. We have given a generic model for examination timetabling problem in Chapter 4.

We have discussed different instances of course and examination timetabling problems which are sub problems of our generalized course and examination timetabling problems in Chapter 5. We have shown that how mathematical models of these instances can be deduced from our generalized examination and course timetabling problem.

We have proposed our algorithms for solving university course and examination timetabling problems in Chapter 6. These are population based algorithms, namely, memetic and honey bee mating algorithm. We have explained their solution representations, their parameters and operators in detail. We have proposed a local search to use in memetic algorithm. We have discussed procedures to make initial solutions for these population based algorithms.

We have analyzed results obtained by our proposed algorithms in Chapter 7. We have applied our memetic algorithm on one benchmark timetabling problem and have compared our results on benchmark instance with other algorithms used in literature. We have also tested our algorithms on generalized educational and examination timetabling problems. We have generated data for both generalized course and examination timetabling problems. We have also solved these datasets with genetic and tabu search algorithms alongside our algorithms and have compared results of these four algorithms while using same termination criteria.

Perspectives

Now, we round off this thesis with some general comments about this work and also give some suggestions for further research work.

We have proposed generic models for timetabling problem which contain many problems as its sub part. These are the problems coming from different departments and we have accumulated them in a single model. When we apply our proposed algorithms on these big size problems, scaling up issues give us clue for future research work. In real world timetabling sometimes, one department courses forms a distinct clump than the courses of other departments. This department may have only few or no common students, may use different campus and may have different set of rooms, with other departments. Thus we think that scheduling of this type of department has little bearing on other departments. So this kind of departments can be scheduled independently by using any relaxation technique while objective is to construct a timetable for whole university. In our opinion, by using this strategy search space can be relaxed and performance of algorithms can be enhanced.

Performance of algorithms may be improved by using any other solution construction processes. Behavior of the crossover operator may be changed by implying some sort of condition on it. For example, we may put restriction on our uniform crossover that it would change a gene only when by changing it timetable remains feasible. Our local search plays a vital role in algorithm and we may use some more grained neighbourhood structures to enhance the performance of algorithm. We have solved course and examination timetabling problems with honey bee mating algorithm. We think, by setting parameters such as population size, number of drones, number of mating flights, size of queen's spermatheca etc. and by using some more sophisticated heuristics to make initial solution can improve performance of the algorithm.

We have also solved our datasets with tabu search algorithm, we may think to incorporate tabu search algorithm with memtic algorithm. Performance of algorithm may be improved by using sets of neighbourhood structures in form of a sequence during the searching process. The tabu list can be used to control the selection of neighbourhood structures, for example if any neighbourhood structure is not improving the result after some specific attempts then algorithm would try next neighbourhood structure in sequence list.
A part of our thesis consists of linear integer programming. We may solve these problems by using lexicographical optimization with sub problems. Preference criterion of cost functions of these sub problems can give edge to prioritize the constraint satisfaction according to the requirements while having broader search space for more preferred objectives.

We have given a transformation of course timetabling to resource constrained project scheduling problem (RCPSP). The use of our memetic and honey bee mating algorithms could be adopted to solve these problems as well. We hope, solution representation of our timetabling algorithms can be converted for RCPSP by concentrating on the given transformation.

REFERENCES

[1] L. Zhipeng and J. K. Hao. Adaptive tabu search for course timetabling, *European Journal of Operational Research* 200, 235-244, 2010.

[2] E. K. Burke, J. Mareceka, A. J. Parkes and H. Rudová. Decomposition, reformulation and diving in university course timetabling, *Computers and Operations Research* 37, 582-597, 2010.

[3] P. D. Causmaecker, P. Demeester and G. V. Berghe. A decomposed metaheuristic approach for a real-world university timetabling problem, *European Journal of Operational Research* 195, 307-318, 2009.

[4] T. A. Duong, V. H. Tam and N. Q. V. Hung. Generating complete university course timetables by using local search methods, *Research, Innovation and Vision for the Future*, 67-74, 2006.

[5] M. Carey. A model and strategy for train pathing with choice of lines, platforms and routes, *Transp. Res.* Part B 28(5), 333-353, 1994.

[6] X. Zhou and M. Zhong. Single-track train timetabling with guaranteed optimality: branch-and-bound algorithms with enhanced lower bounds, *Transp. Res.* Part B 41(3) 320-341, 2007.

[7] O. Guyon, P. Lemaire, É. Pinson and D. Rivreau. Cut generation for an integrated employee timetabling and production scheduling problem, *European Journal of Operational Research* 201, 557-567, 2010.

[8] L. Zhipeng and J. K. Hao. Adaptive neighborhood search for nurse rostering, *European Journal of Operational Research* 218, 865-876, 2012.

[9] A. Viana and J. S. Pinho. Using metaheuristics in multiobjective resource constrained project scheduling, *European Journal of Operational Research* 120, 359-374, 2000.

[10] A. Schaerf. A survey of automated timetabling, *Artificial Intelligence Review* 13(2), 87-127, 1999.

[11] D. Corne, P. Ross, and H. Fang. Evolving timetables in the practical handbook of genetic algorithms, L. C. Chambers (Ed.), *CRC Press*, 1, 219-276, 1995.

[12] S. Daskalaki , T. Birbas and E. Housos. An integer programming formulation for a case study in university timetabling, *European Journal of Operational Research* 153, 117-135, 2004.

[13] A. Pasquale and I. Vasil'ev. A computational study of a cutting plane algorithm for university course timetabling, *Journal of Scheduling* 8, 497-514, 2005.

[14] G. Kendall, S. Knust, C. C. Ribeiro and S. Urrutia. Scheduling in sports: An annotated bibliography, *Computers and Operations Research* 37, 1-19, 2010.

[15] S. Abdullah, E. K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem, *Metaheuristics, Operations Research / Computer Science Interfaces Series* 39, 153-169, 2007.

[16] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid approach for the university course timetabling problem, *Journal of Scheduling* 9(5), 403-432, 2006.

[17] L. D. Gaspero and A. Schaerf. Neighborhood portfolio approach for local search applied to timetabling problems, *Journal of Mathematical Modeling and Algorithms* 5(1), 65-89, 2006.

[18] P. Kostuch. The university course timetabling problem with a three-phase approach, *Practice and Theory of Automated Timetabling* (*PATAT 2004*), 109-125, 2004.

[19] A. Bonutti, F. D. Cesco, L. D. Gaspero and A.Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results, *Annals of operations research*, 194(1), 59-70, 2012.

[20] D. Werra. Graphs, hypergraphs and timetabling, *Methods of Operations Research* 49, 201-213, 1985.

[21] E. K. Burke, J. Kingston and D. Werra. Applications to timetabling. In: J. Gross, J. Yellen (Eds.), Handbook of Graph Theory, *Chapman Hall/CRC Press*, 445-474, 2004.

[22] P. Brucker, A. Drexl, R. Mohring, K. Neumannd and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods, *European Journal of Operational Research* 112, 3-41, 1999.

[23] K. Fleszar and K. S. Hindi. Solving the resource-constrained project scheduling problem by a variable neighbourhood search, *European Journal of Operational Research* 155 402-413, 2004.

[24] S. Daskalaki and T. Birbas. Efficient solutions for a university timetabling problem through integer programming, *European Journal of Operational Research* 160, 106-120, 2005.

[25] M.Ahmad, C. Caux and M. Gourgand. Generic model for university course timetabling problem solved by genetic algorithm, *Industrial Simulation Conference* (*ISC 2012*), 4-6 June, Brno, Czech Republic, 209-216, 2012.

[26] A. F. Jacques and A. Lavoie. Exchanges procedures for timetabling problems, *Discrete Applied Mathematics* 35, 237-253, 1992.

[27] N. Boland, B. D. Hughes, L. T. G. Merlotb and P. J. Stuckey. New integer linear programming approaches for course timetabling, *Computers and Operations Research* 35, 2209-2233, 2008.

[28] S. Daskalaki, T. Birbas and E. Housos. An integer programming formulation for a case study in university timetabling, *European Journal of Operational Research* 153, 117-135, 2004.

[29] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update, *European Journal of Operational Research* 174, 23-37, 2006.

[30] K.J. Batenburg and W.J. Palenstijn, A new exam timetabling algorithm, *Belgian-Dutch Artificial Intelligence Conference* (*BNAIC 2003*), 19-26, 2003.

[31] J. M. Thompson and K. A. Dowsland. A robust simulated annealing based examination timetabling system, *Computers and Operations Research* 25(7-8), 637-648, 1998.

[32] C. Beyrouthy, E. K. Burke, D. Landa-Silva, M. B. Collum, M. P. Mullan and A. J. Parkes. Towards improving the utilisation of university teaching space, *Journal of the Operational Research Society* 60(1), 130-143, 2009.

[33] E. K. Burke, J. C. Mare, A. J. Parkes and H. Rudová. On a clique-based integer programming formulation of vertex colouring with applications in course timetabling, *Technical Report NOTTCS-TR-2007-10, The University of Nottingham*, Nottingham, 2007.

[34] P. Ross, E. Hart, and D. Corne. In Advances in Evolutionary Computing: Theory and Applications, topic: Genetic algorithms and timetabling, A. Ghosh and K. Tsutsui, (Eds.), *Springer-Verlag*, New York, 755- 771, 2003.

[35] T. Müller. ITC2007 solver description: A hybrid approach, *Practice and Theory of Automated Timetabling* (*PATAT 2008*), 2008.

[36] M.J. Geiger. An application of the threshold accepting metaheuristic for curriculum based course timetabling, *Practice and Theory of Automated Timetabling* (*PATAT 2008*), 2008.

[37] M. Clark, M. Henz, B. Love and F. Quik. A repair-based timetable solver, *Practice and Theory of Automated Timetabling* (*PATAT 2008*), 2008.

[38] S. Abdullah, H. Turabieh, B. McCollum and E. K. Burke. An investigation of a genetic algorithm and sequential local search approach for curriculum-based course timetabling problems, *Multidisciplinary International Conference on Scheduling: Theory and Applications* (*MISTA 2009*), 10-12 August, Dublin, Ireland, 2009.

[39] K. Shaker and S. Abdullah. Incorporating great deluge approach with kempe chain neighbourhood structure for curriculum-based course timetabling problems, *Data Mining and Optimization (DMO 2009*), 149-153, 2009.

[40] B. McCollum, A. Schaerf, B. Paechter, P.McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, Q. Rong and E. K. Burke. Setting the research agenda in automated timetabling, *The Second International Timetabling Competition INFORMS Journal on Computing* 22(1), 120-130, 2010.

[41] L. Zhipeng, J. K. Hao and F. Glover. Neighborhood analysis: A case study on curriculum based course timetabling, *Technical Report, LERIA, University of Angers*, France, 2009.

[42] http://www.cs.qub.ac.uk/itc2007/

[43] L. D. Gaspero, B. McCollum and A. Schaerf. The second international timetabling competition (ITC-2007): curriculum-based course timetabling (Track 3), in: *International Workshop on Scheduling a Scheduling Competition, International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 22–26 September, Providence, Rhode Island, USA, 2007.

[44] B. McCollum, P. McMullan, A.J. Parkers, E.K. Burke and R. Qu. A new model for automated examination timetabling, *Annals of Operations Research* 194(1), 291-315, 2012.

[45] M. Alzaqebah and S. Abdullah. Artificial bee colony search algorithm for examination timetabling problems, *International Journal of the Physical Sciences* 6(17), 4264-4272, 2011.

[46] K. Nguyen, P. Nguyen and N. Tran. A hybrid algorithm of harmony search and bees algorithm for a university course timetabling problem, *IJCSI International Journal of Computer Science Issues* 9(1), 2012.

[47] M. Alzaqebah and S. Abdullah. The bees algorithm for examination timetabling problems, *International Journal of Soft Computing and Engineering (IJSCE)* 1(5), 2231-2307, 2011.

[48] N. R. Sabar, M. Ayob, G. Kendall and Q. Rong. A honey-bee mating optimization algorithm for educational timetabling problems, *European Journal of Operational Research* 216, 533-543, 2012.

[49] A. Baykasoulu, L. Ozbakır, P. Tapkan. Artificial bee colony algorithm and its application to generalized assignment problem, In: T.S.C. Felix and M. K. Tiwari (Eds.), Swarm Intelligence, Focus on Ant and Particle Swarm Optimization, *I-Tech Education and Publishing*, Vienna, Austria, ISBN 978-3-902613-09-7, page 532, 2007.

[50] E. K. Burke, J. P. Newall and R. F. Weare. A memetic algorithm for university exam timetabling, In: E. K. Burke, P. Ross (Eds.), *Practice and Theory of Timetabling* (*PATAT 1995*), *Lecture Notes in Computer Science*, *Springer, Heidelberg*, 1153, 241-250, 1996.

[51] K. Socha, M. Samples and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art, *In: Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003), Lecture Notes in Computer Science* 2611, 334-345, *Springer-Verlag*, Berlin, 2003.

[52] N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetable problem, *Applied Soft Computing* 10, 457-467, 2010.

[53] P. Cote, T. Wong, R. Sabourin. A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem, In: E. K. Burke, M. Trick (Eds.), *Practice and Theory of Automated Timetabling (PATAT 2004), Lecture Notes in Computer Science* 3616, 294-312, *Springer*, 2005.

[54] M. Eley. Ant algorithms for the exam timetabling problem, In: E. K. Burke, H. Rudova (Eds.), *Practice and Theory of Timetabling (PATAT 2006), Lecture Notes in Computer Science, Springer, Heidelberg*, 3867, 364-382, 2007.

[55] D. Costa and A. Hertz. Ant can colour graphs, *Journal of Operational Research Society* 48, 295-305, 1997.

[56] E. Ersoy, E. Ozcan and A. S. Etaner. Memetic algorithms and hyper hill-climbers, *Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007)*, 159-166, 2007.

[57] E. K. Burke, A. J. Eckersley, B. McCollum, S. Petrovic and R. Qu. Hybrid variable neighbourhood approaches to university exam timetabling, *European Journal of Operational Research* 206, 46-53, 2010.

[58] D. Landa-Silva and J. H. Obit. Evolutionary non-linear great deluge for university course timetabling, Hybrid Artificial Intelligent *(HAIS 2009)* LNAI, *Springer-Verlag*, Berlin, Heidelberg 5572, 269-276, 2009.

[59] S. Abdullah, E. K. Burke, and B. McCollum. A hybrid evolutionary approach to the university course timetabling problem, IEEE Congress on Evolutionary Computation *(CEC2007)*, 1764-1768, 2007.

[60] H. Turabieh, S. Abdullah, and B. McCollum. Electromagnetism-like mechanism with force decay rate great deluge for the course timetabling problem, *Rough Set and Knowledge Technology (RSKT 2009), In: P. Wen et al. (Eds.),Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg*, 5589, 497-504, 2009.

[61] K. V. Frisch. Bees: Their Vision, Chemical Senses and Language, *Cornell University Press*, New York, Ithaca, 1976.

[62] T. D. Seeley. The wisdom of the hive: The social physiology of honey bee colonies, *Massachusetts: Harvard University Press*, Cambridge, 1996.

[63] E. Bonabeau, M. Dorigo and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems, *Oxford University Press*, New York, 1999.

[64] S. Camazine, J. Deneubourg, N. R. Franks, J. Sneyd, G. Theraula and E. Bonabeau. Self-Organization in Biological Systems, Princeton: *Princeton University Press*, 2003.

[65] X. T. Ioannis and G. N. Beligiannis. A hybrid particle swarm optimization based algorithm for high school timetabling problems, *Applied Soft Computing* 12(11), 3472-3489, 2012.

[66] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim and M. Zaidi. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.

[67] D.T. Pham, E. Kog, A. Ghanbarzadeh, S. Otri, S. Rahim, M. Zaidi. The Bees Algorithm - A novel tool for complex optimisation problems, *Intelligent Production Machines and Systems* (IPROMS 2006), Oxford, 2006.

[68] H. A. Abbass. A monogenous MBO approach to satisfiability, *Computational Intelligence for Modeling, Control and Automation* (CIMCA 2001), Las Vegas, NV, USA, 2001.

[69] H. A. Abbass. Marriage in honey-bee optimization (MBO): A haplometrosis polygynous swarming approach, IEEE Congress on Evolutionary Computation (*CEC 2001)*, Seoul, Korea, 207-214, 2001.

[70] S. J. Naseem and S. Yang. A guided search genetic algorithm for the university course timetabling problem, *Multidisciplinary International Conference on Scheduling: Theory and Applications* (MISTA 2009) 10-12 August, Dublin, Ireland, 2009.

[71] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling, *Journal of Scheduling* 9(5) 403-432, 2006.

[72] S.A. M. Hassani. A computational approach to enhancing course timetabling with integer programming, *Applied Mathematics and Computation* 175, 814-822, 2006.

[73] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling, *European Journal of Operational Research* 140, 266-280, 2002.

[74] S. Massoodian and A. Esteki. A Hybrid Genetic Algorithm for Curriculum Based Course Timetabling, *Practice and Theory of Automated Timetabling (PATAT 2008)*, 18-22 August, Montreal, Canada.

[75] Z. Bratkovic, T. Herman, V. Omrcen, M. Cupic and D. Jakobovic. University course timetabling with genetic algorithm: a laboratory exercises case study, *Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science* 5482, 240-251, 2009.

[76] E. Özcan and A. Alkan. A memetic algorithm for solving a timetabling problem: An incremental strategy, In: P. Baptiste, G. Kendall, A.M. Kordon and F. Sourd (Eds.), *Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007),* 394-401, 2007.

[77] O. Rossi-Doria and B. Paechter. A memetic algorithm for University Course Timetabling, In *Combinatorial optimisation 2004*, Book of abstracts page, 56.

[78] M. Chiarandini, M. Birattari, K. Socha and O. Rossi-Doria. An effective hybrid approach for the University Course Timetabling Problem, Journal of Scheduling 9(5), 403-432, 2006.

[79] K. Socha. The influence of run-time limits on choosing ant system parameters, *International Conference on Genetic and evolutionary computation, (GECCO 2003),* 49-60, 2003.

[80] H.M. Terashima, P. Ross and M.R. Valenzuela. Evolution of constraint satisfaction strategies in examination timetabling, *genetic and evolutionary computation conference* (GECCO), Morgan Kauffmann, 635-642, 1999.

[81] D. G. Whitley. A Different Genetic Algorithm, *Rocky Mountain Conference on Artificial Intelligence*, Denver, USA, 1988.

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

[83] O. Rossi-Doria and Ben Paechter. An hyperheuristic approach to course timetabling problem using an evolutionary algorithm, *Technical Report, Napier University,* Edinburgh, Scotland, 2003.

[84] C. Blum, S. Correia, M. Dorigo, B. Paechter, O. Rossi-Doria and M. Snoek. A GA evolving instructions for a timetable builder, *Practice And Theory of Automated Timetabling* (PATAT 2002), Gent, Belgium, 120-123, 2002.

[85] E. Ozcan, A. J. Parkes and A. Alkan. The interleaved constructive memetic algorithm and its application to timetabling, *Computers and Operations Research* 39, 2310-2322, 2012.

[86] H. Kanoh and Y. Sakamoto. Knowledge-based genetic algorithm for university course timetabling problems, *International Journal of Knowledge-based and Intelligent Engineering Systems* 12(4), 283-294, 2008.

[87] S. Abdullah and H. Turabieh. On the use of multi neighbourhood structures within a tabu based memetic approach to university timetabling problems, *Information Sciences* 191, 146-168, 2012.

[88] R. M. Santiago, S. S. Salcedo, M. D. C. Prado and C. C. Bousono. A two-phase heuristic evolutionary algorithm for personalizing course timetables: a case study in a Spanish university, *Computers and Operations Research* 32, 1761-1776, 2005.

[89] A. Jula and N. N. Khatoon. Using CMAC to obtain dynamic mutation rate in a metaheuristic memetic algorithm to solve university timetabling problem, *European journal of scientific research* 63(2), 172-181, 2011.

[90] R. Qu and E. K. Burke. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems, *Journal of the Operational Research Society* 60, 1273-1285, 2009.

[91] F. Makoto. A hybrid algorithm for the university course timetabling problems, *Journal of Japan Society for Fuzzy Theory and Intelligent Informatics* 22(1), 142-147, 2010.

[92] G. Dueck. New optimization heuristics the great deluge algorithm and the record-to-record travel, *Journal of Computational Physics* 104(1), 86-92, 1993.

[93] D. F. Shiau. A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences, *Expert Systems with Applications* 38, 235-248, 2011.

[94] D. Datta, D. Kalyanmoy and C.M. Fonseca. Solving class timetabling problem of IIT Kanpur using multi-objective evolutionary algorithm, *KanGAL Report Number 2006006.*

[95] M.Nandhini, S.Kanmani and S.Anandan. Performance analysis of diversity measure with crossover operators in genetic algorithm, *International Journal of Computer Applications* 19 (2), 19-26, 2011.

[96] R. Qu, E. K. Burke, B. McCollum, L.T.G. Merlot and S. Y. Lee. A survey of search approaches and automated system development for examination timetabling, *Journal of Scheduling* 12(1), 55-89, 2009

[97] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems, *OR Spectrum* 30, 167-190, 2008.

[98] N. R. Sabar, M. Ayob and G. Kendall. Tabu exponential Monte-Carlo with counter heuristic for examination timetabling, IEEE Symposium on Computational Intelligence in Scheduling (CISched 2009), Nashville, Tennessee, USA, 90-94, 2009.

[99] N. R. Sabar, M. Ayob, G. Kendall and R. Qu. Roulette wheel graph colouring for solving examination timetabling problems, *Combinatorial Optimization and Applications*, *Lecture Notes in Computer Science*, Springer, Berlin 5573, 463-470, 2009.

[100] M. Ayob, A. M. A. Malik, S. Abdullah, A.R. Hamdan, G. Kendall and R. Qu. Solving a practical examination timetabling problem: a case study, In: O. Gervasi, M. Gavrilova: (Eds.), (ICCSA 2007), Part III, *Lecture Notes in Computer Science*, *Springer*, Heidelberg 4707, 611-624, 2007.

[101] S. Abdullah and H. Turabieh. Generating university course timetable using genetic algorithm and local search, *Hybrid Information Technology*, 254-260, 2008.

[102] R. Lewis. Metaheuristics For University Course Timetabling, *Ph.D* Thesis, 2006.

[103] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning, *Addison-Wesley Publishing Company*, 1989.

[104] B.J. Reardon. Fuzzy logic versus niched pareto multiobjective genetic algorithm optimization, Modeling and Simulation in Material Sciences and Engineering 6, 717-735, 1998.

[105] Z. Bingul, A.S. Sekmen and S. Zein-Sabatto. Evolutionary approach to multiobjective problems using adaptive genetic algorithms, *Systems, Man, and Cybernetics*, October 8-11, Nashville, TN, 2000.

[106] Z. Bingul, A.S. Sekmen and S. S. Zein. Genetic algorithms applied to real time multiobjective optimization problems, *Southeast Congress Conference* (SoutheastCON'2000), April, Nashville, TN, USA, 95-103, 2000.

[107] Z. Bingul. Adaptive genetic algorithms applied to dynamic multiobjective problems efficient combination of genetic operators, *Applied Soft Computing* 7, 791-799, 2007.

[108] P.K. Chawdhry. Soft computing in engineering design and manufacturing, London: *Springer*, ISBN 3-540-76214-0, 1998.

[109] U. Bodenhofer. Genetic Algorithms: Theory and Applications, Lecture Notes Second Edition, 2000.

[110] M. Mitchell. An Introduction to Genetic Algorithms. *A Bradford Book The MIT Press*, 1998.

[111] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs, Second Edition. *Springer*, 1994.

[112] B. Chakraborty and P. Chaudhuri. On the use of genetic algorithm with elitism in robust and nonparametric multivariate analysis, *Austrian journal of statistics* 32(1-2), 13-27, 2003.

[113] F. D. Qarouni, N. A. Ardabili and M. H. Moeinzadeh**.** Finding feasible timetables with particle swarm optimization, *Innovations in Information Technology (IIT 2007),* 387-391, 2007.

[114] N. J. Sadaf and S. Yang. A Memetic Algorithm for the University Course Timetabling Problem, *International Conference on Tools with Artificial Intelligence, (ICTAI 2008)* 1, 427-433, 2008.

[115] R. Tavares, A. Teófilo, P. Silva and A. C. Rosa. Infected genes evolutionary algorithm, *Proceedings of the 1999 ACM symposium on applied computing*, 333-338, 1999.

[116] D. Brelaz. New methods to color the vertices of a graph, *Communications of the ACM* 22(4), 251-256, 1979.

[117] A.Wren. Scheduling, timetabling and rostering - A special relationship, *Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science* 1153, 46-75, 1996.

[118] D.Werra. An introduction to timetabling, *European journal of Operational research*, 19, 151-162, 1985.

[119] M.Ahmad, C. Caux, M. Chabrol and M. Gourgand. A survey on recent developments in automated timetabling, *12^{e} congrès annuel de la Société française de Recherche Opérationnelle et d'Aide à La Décision (ROADEF 2011)*, 2-4 March, Saint-Etienne, France 2, 833-834, 2011.

[120] J. A. Breslaw. A linear programming solution to the faculty assignment problem, *Socio-Economic Planning Science* 10, 227-230, 1976.

[121] K. Schimmelpfeng and S. Helber. Application of a real world university course timetabling model solved by integer programming, *OR Spectrum* 29, 783-803, 2007.

[122] T. Birbas, S. Daskalaki and E. Housos. Timetabling for Greek high schools, *Journal of the Operational Research Society* 48, 1191-1200, 1997.

[123] A. Mingozzi, V. Maniezzo**,** S. Ricciardelli and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation, *Management Science* 44, 714-729, 1998.

[124] F. Ballestın,V. Valls and S. Quintanilla. Pre-emption in resource-constrained project scheduling, *European Journal of Operational Research* 189, 1136-1152, 2008.

[125] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem, *European Journal of Operational Research* 207, 1-14, 2010.

[126] S. Hartmann and R. Kolisch. Experimental evaluation of the state of the art heuristics for the resource constrained project scheduling problem, *European Journal of Operational Research* 127, 394-407, 2000.

[127] M. Ahmad, M. Gourgand and C. Caux. Transformation of course timetabling problem to RCPSP, *World Academy of Science, Engineering and Technology* 168, 2192-2197, 2012.

[128] R. Qu, E.K. Burke, B. McCollum and L.T.G. Merlot, A survey of search methodologies and automated system development for examination timetabling, *Journal of Scheduling* 12, 55-89, 2009.

[129] M. Carter. A survey of practical applications of examination timetabling Algorithms, *Operations Research* 34, 193-202, 1986.

[130] M. Carter and G. Laporte, Recent developments in practical examination timetabling, *Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science,* E. Burke and P. Ross (Eds.), *Springer-Verlag*, Berlin 1153, 1-21, 1996.

[131] E. Burke, D. Elliman, and R. Weare. The automation of the timetabling process in higher education, *Journal of Education Technology Systems* 23, 257- 266, 1995.

[132] E. K. Burke, D. G. Elliman, P. H. Ford, and R. Weare. Examination timetabling in british universities: A Survey, *Practice and Theory of Automated Timetabling (PATAT 1995), Lecture Notes in Computer Science*, E. Burke and P. Ross (Eds.), Springer-Verlag, 1153, 76-92, 1996.

[133] M. Carter and G. Laporte. Recent developments in practical course timetabling. *Practice and Theory of Automated Timetabling (PATAT 1997), Lecture Notes in Computer Science*, E. Burke and M. Carter (Eds.), Springer-Verlag, Berlin 1408, 3-19, 1998.

[134] A. Tripathy. School timetabling - A case in large binary linear integer programming, *Managment Science* 30, 1473-1489, 1984.

[135] M. Carter. A langarian relaxation approach to the classroom assignment problem, *INFOR* 27, 230-246, 1986.

[136] J. Broek, C. Hurkens, G. Woeginger. Timetabling problems at the TU Eindhoven, *European Journal of Operational Research* 196(3), 877-885, 2009.

[137] S. M. Yakoob and H. D. Sherali. Mathematical programming models and algorithms for a class-faculty assignment problem, *European Journal of Operational Research* 173(2), 488-507, 2006.

[138] S. M. Yakoob and H. D. Sherali. A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations, *European Journal of Operational Research* 180(3), 1028-1044, 2007.

[139] R.H. McClure and C.E. Wells. A mathematical programming model for faculty course assignment, *Decision Science* 15, 409-420, 1984.

[140] S. Kirkpatrick, C. D. Gellatt and M. P. Vecchi. Optimization by simulated annealing, *Science*, 220, 671-680, 1983.

[141] R. klein. Scheduling of resource-constrained projects, *Kluwer academic publishers*, 2000.

[142] B. Bullnheimer. An examination scheduling model to maximize students' study time, E. K. Burke, M. W. Carter (Eds.), *Practice and Theory of Automated Timetabling (PATAT 1997), Lecture Notes in Computer Science, Springer*, New York, 1408, 78-91, 1998.

[143] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms, *Management Science* 37, 98-113, 1991.

[144] F. Melicio and J. Caldeira. Implementation aspects of simulated annealing on timetabling, *IEEE Systems Science and Systems Engineering*, Beijing. Aceite, 1998.

[145] S. Elmohamed, G. Fox, and P. Coddington. A comparison of annealing techniques for academic course scheduling, *Practice and Theory of Automated Timetabling (PATAT 1997), Lecture Notes in Computer Science*, E. Burke and M. Carter (Eds.) Springer-Verlag, Berlin 1408, 146-166, 1998.

[146] D. Zhang, Y. Liu, R. M'Hallah and S. C. H. Leung. A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems, *European Journal of Operational Research* 203(3), 550-558, 2010.

[147] S. Ceschia, L.D. Gaspero and A. Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem, *Computers and Operations Research* 39(7), *1615-1624, 2012.*

[148]A. Gunawan, K.N. Ming and K. L. Poh. A hybridized lagrangian relaxation and simulated annealing method for the course timetabling problem, Computers and Operations Research 39(12), 3074-3088, 2012.

[149] K. Socha, J. Knowles, and M. Samples. A MAX-MIN Ant system for the university course timetabling problem, *Third International Workshop on Ant Algorithms (Ants 2002), Lecture Notes in Computer Science*, M. Dorigo, G. Di Caro, and M. Samples (Eds.), *Springer-Verlag*, Berlin 2463, 1-13, 2002.

[150] K. A. Dowsland and J. Thompson. Ant colony optimization for the examination scheduling problem, *The Journal of the Operational Research Society* 56(4), 426-438, 2005.

[151] C. Nothegger, A. Mayer, A. Chwatal, G. R. Raidl. Solving the post enrolment course timetabling problem by ant colony optimization, *Annals of Operations Research* 194(1), 325-339, 2012.

[152] M. Eley. Some experiments with ant colony algorithms for the exam timetabling problem, *Ant Colony Optimization and Swarm Intelligence* (ANTS 2006), 492-499, 2006.

[153]E.K. Burke, J.P. Newall and R.F. Weare. Initialization strategies and diversity in evolutionary timetabling, *Evolutionary computation* 6(1), 81-103, 1998.

[154] L.T.G. Merlot, N. Boland, B.D. Hughes and P.J. Stuckey. A hybrid algorithm for the examination timetabling problem. *In: E.K. Burke and P. D. Causmaecker (Eds.) Practice and Theory of Automated Timetabling (PATAT 2002), Lecture Notes in Computer Science* 2740, 207-231, 2003.

[155] T.H. Hultberg and D.M. Cardoso. The teacher assignment problem: A special case of the fixed charge transportation problem, *European Journal of Operational Research* 101, 463-473, 1997.

[156] M.A. Badri, D.L. Davis, D.F. Davis and J. Hollingsworth. A multi-objective course scheduling model: Combining faculty preferences for courses and times, *Computers and Operations Research* 25(4), 303-316, 1998.

[157] K. Gosselin and M. Truchon. Allocation of classrooms by linear programming, *Journal of Operational Research Society* 37(6), 561-569, 1986.

[158] A. Tripathy. School timetabling-A case in large binary integer linear programming, *Management Science* 30(12), 1473-1489, 1984.

[159]M. R. Garey and D. S. Johnson, Computers and Intractability: A guide to NPcompleteness, First Ed. San Francisco: *W. H. Freeman and Company*, 1979.

[160] C.H.Aladag and G.Hocaoglu. A tabu search algorithm to solve a course timetabling problem, *Hacettepe Journal of Mathematics and Statistics* 36(1), 53-64, 2007.

[161]P. Pongcharoen, W. Promtet, P. Yenradee and C. Hicks. Stochastic optimisation timetabling tool for university course scheduling, *International Journal of Production Economics* 112, 903-918, 2008.

[162] M. Sagir and Z. K. Ozturk. Exam scheduling: Mathematical modeling and parameter estimation with the analytic network process approach, *Mathematical and Computer Modelling* 52, 930-94, 2010.

[163] P. Boizumault, Y. Delon and L. Peridy. Constraint logic programming for examination timetabling, *The Journal of Logic Programming* 26(2), 217-233, 1996.

[164] M. Ayob, A. H. Razak, S. Abdullah et al. Intelligent examination timetabling software, *Procedia Social and Behavioral Sciences* 18, 600-608, 2011.

[165] G. M. White, B. S. Xie and S. Zonjic. Using tabu search with longer-term memory and relaxation to create examination timetables, *European Journal of Operational Research* 153, 80-91, 2004.

[166] P. Brucker and S. Knust. Resource-constrained project scheduling and timetabling. E. Burke and W. Erben (Eds.) *Practice and Theory of Automated Timetabling* (PATAT 2000), *Lecture Notes in Computer Science, Springer-Verlag* Berlin Heidelberg 2079, 277-293, 2001.

[167] R. V. Alvarez, E. Crespo and J. M. Tamarit. Design and implementation of a course scheduling system using tabu search, *European Journal of Operational Research* 137, 512-523, 2002.

[168] C. H. Aladag, G. Hocaoglu and M. A. Basaran. The effect of neighbourhood structures on tabu search algorithm in solving course timetabling problem, *Expert systems with applications* 36, 12349-12356, 2009.