AUTOMATION OF TIME TABLE GENERATION USING GENETIC ALGORITHM

Sonal D Shenoy¹, Vaibhav Sharma¹, S.Santhanalakshmi¹
sonalshenoy@gmail.com
vaibhav230292@gmail.com
s_lakshmi@blr.amrita.edu
¹Department of CSE, Amrita School of Engineering, Bengaluru

Abstract

Finding a feasible lecture/tutorial timetable in a large university department is a challenging problem faced continually in educational establishments. This paper presents an evolutionary algorithm (EA) based approach to solving a heavily constrained university timetabling problem. The approach uses a problem-specific chromosome representation. Heuristics and context-based reasoning have been used for obtaining feasible timetables in a reasonable computing time. An intelligent adaptive mutation scheme has been employed for speeding up the convergence. The comprehensive course timetabling system presented in this paper has been validated, tested and discussed using real world data from a large university.

Keywords – Genetic Algorithms, Timetabling, Scheduling, and Automated.

I. INTRODUCTION

Scheduling is the task of allocation, arrangement or distribution of certain objects, subject to specific constraints, in a pattern of time or space [1]. The aim of such an arrangement may be optimal allocation of available resources in order to achieve minimization of time or cost. Alternately, the aim of a scheduling problem may only be to obtain a feasible solution based on the constraints, since in certain scheduling problems, all feasible solutions have equal cost and are optimal. This is the case with the Timetabling problem dealt with in this paper.

Objects in a scheduling problem may be workers, vehicles, machines, etc. Constraints are the rules that govern the process of scheduling (such as restrictions on time or available resources), and certain constraints may be inviolable for the problem under consideration, while others take the form of principles that should, but need not necessarily be satisfied. A Genetic Algorithm (GA) is a parallel search algorithm based on the Darwinian evolutionary theory [2]. It is used to search a large solution space which may be discontinuous or nonlinear, and/ or where expert knowledge is lacking or difficult to encode. Genetic algorithms form part of a larger category of algorithms called Evolutionary Algorithms (EA), which apply techniques inspired by natural evolution- such as mutation, selection, and crossover, and use them to generate solutions to optimization problems.

The process of Selection in Genetic Algorithms follows the Darwinian principle of Survival of the Fittest, by allotting a large number of copies in the following generations to fitter chromosomes, and allotting none or few copies to unfit chromosomes. This makes the gene pool, on average, much fitter. The process of crossover involves exchange of genetic information between two randomly chosen mates, which goes on to produce one or multiple offspring, which may be distinct from their parent chromosomes. The process of mutation involves random change of allele values in a chromosome, in order to bring about genetic variation.

The time-tableing problem (TTP) is basically the scheduling and assignment of the lessons into appropriate time-slots and resources respectively, without causing time clashes for the students and the teachers, as well as the resource clashes [3]. The drawing up of university timetables is a slow, laborious task, performed by people working on the strength of their knowledge of resources and constraints of a specific institution.

The construction of course timetables for academic institutions is a very difficult problem with a lot of constraints that have to be respected and a huge search space to be explored, even if the size of the problem input is not significantly large, due to the exponential number of the possible feasible timetables. On the other hand, the problem itself does not have a widely approved definition, since different variations of it are faced by different departments. This problem has therefore proven to be a very complex and time-consuming problem. Timetables are considered feasible provided the so-called hard constraints are respected. However, to obtain high-quality timetabling solutions, soft constraints, which impose satisfaction of a set of desirable conditions for classes and teachers should be satisfied.

Over the past few years, a wide array of techniques have been proposed for solving the course timetabling problem and its’ variants. Several techniques have been developed for automated timetable generation [4, 5]. Each implementation, however, is specific to the academic institution for which it is
developed, and addresses a different problem with diverse constraints.

Although, the timetabling problem is distinctive for each institution, there exist a set of entities and constraints which are common to every possible instantiation of the timetabling problem. In this paper, we present a model of this common core in terms of a knowledge-augmented evolutionary algorithm approach which may be extended to cover the needs of a specific academic unit. The algorithm incorporates two distinct objectives that minimization of the violations of both types of constraints, hard and soft. The problem is modeled as a bi-objective problem to construct feasible assignments of course modules to lecture rooms on specified timeslots. A new representation for the timetabling problem is presented.

A brief description of the problem is presented, followed by the evolutionary algorithm, using a standard fitness-sharing scheme improved with an elitist secondary population. This approach represents each timetabling solution with a matrix-type chromosome and is based on special-purpose genetic operators of crossover and mutation developed to act over the population. The paper concludes with a discussion of the favorable results obtained through an application of the algorithm to a real instance taken from a university.

II. THE TIME-TABLING PROBLEM
The time tabling problem can be modeled as a constraint satisfaction problem with many parameters and loose constraints. These constraints have to be modeled in a format that can be handled efficiently by the scheduling algorithm. The constraints considered for this problem are:

The hard constraints: These are constraints that must be satisfied in order to obtain a feasible timetable. The following constraints would be in violable for a University Time Table Scheduling problem:

1. No room can have more than one scheduled class in a given time slot.
2. No student group can have more than one scheduled class in a given time slot.
3. No faculty member can have more than one scheduled class at a given time slot.
4. The number of classes per week is a fixed number for each subject, and this number must be met by the prepared schedule.
5. Lab classes must be in consecutive hours.

The soft constraints: These are constraints of lower priorities to be satisfied. The violation of the soft constraints will not cause the timetable to lose its feasibility. Examples of soft constraints are:

1. There cannot be more than 2 classes for a subject on one day.
2. There cannot be too many empty slots in between lessons.

The objective of the optimization algorithm is to obtain the best schedule while satisfying all these constraints.

III. PROPOSED SYSTEM
The proposed system has ‘L’ no of lecturers, ‘S’ no of subjects and ‘C’ no of classes per subject ‘i’ per week. Each day has ‘H’ no of hours and we have five working days per week. The total no. of time slots then become equal to 5*H. The problem then becomes assigning \[C_1 + C_2 + \ldots + C_n\] number of classes in the ‘5*H’ time slots.

IV. GA IMPLEMENTATION
A program in java was developed which employed GA methods to perform Automated Timetabling. Consequently, the program was entitled “gaatt.java”. The GA operates upon a population of timetables, which are maintained in memory. Each timetable is evaluated by testing the number of times it breaches each constraint. Thus timetables are evolved with a minimum number of constraint violations. A detailed discussion of the encoding scheme and the GA operators used is given in the following subsections:

A. Chromosome Representation
The chromosome is made up of genes and each gene is, in itself, a time table. This means that every gene would carry a large amount of data- regarding subjects, faculty, and rooms allotted to each slot for each student group in a week. The structure of the gene would be 5 x 6 matrix, since it must represent a time table for a 5- day week, with 6 allotted time slots per day.

Each time slot contains the information regarding the faculty code, subject code, and room code where that class would be held. Each gene would be a 5 x 6 array of a single time slot. The chromosome would be an array of genes.

B. Initialization
The initialization of the algorithm is split into 2 parts: Initialization of the data and the random generations of the initial chromosome population.

Initialization of data: The data that is required by the program are as follows:
- Total number of classes.
- The number of timeslots needed for lecture per week.
- Assign appropriate faculty to the course.

C. Repair Strategy
The hard constraints are discussed below:

Hard constraint 1, 2, and 3: neither a class nor a teacher nor a room is assigned to more than one lesson in the same period.

[Analysis and Strategy]: To ensure that this constraint is not violated, we need to examine whether or not a particular teacher/student/room is assigned more than once. This is accomplished by simply checking vertically the keyword lecturer/class in each time slot, as shown in Fig. 1.
When hard constraints violations are detected, the chromosome should be repaired to be feasible. One way of doing this is to move the infeasible events horizontally, within the size range to which they belong by checking constantly to avoid conflicts, as shown in Fig. 2.

Hard constraint 1: The number of classes per week is a fixed number for each subject, and this number must be met by the prepared schedule.

[Analysis and Strategy]: Pseudo code to assure exactly $C_i$ no. of classes per subject per week is provided in Fig. 3.

For each subject:
Set the count to 0
For each time slot which is assigned a subject:
Add 1 to the subject count
If subject occurs more than specified credits:
Replace the first occurrence of that subject in the time slot with the subject whose occurrence is less than the specified credits given its occurrence are zero on that particular day.

Hard constraint 5: Lab classes must be in consecutive hours.

[Analysis and Strategy]: The first lesson of lab is followed by the second. We also need the first of the double periods not to be assigned to the last period of each day. One way of doing this to move all lessons $1->2, 2->3, 3->4, 4->5, 5->6,$ and $6->1.$

Soft constraint 1: There cannot be more than 2 classes for a subject on one day.
[Analysis and Strategy]: Any classes which appear more than once are altered such that they appear only once, as outlined in Fig. 4. Any classes which do not appear at all are booked to a spare space as outline in Fig. 5.

For each working day:
   For each subject:
      Set the count to 0.
      For each time slot which is assigned a subject:
         Add 1 to the subject count.
      If a subject occurs more than once then
         Choose the first occurrence of the subject
         turn it to a free slot.

Fig. 4 – Pseudo Code for First Stage of Repair Strategy for soft constraint 1

For each working day:
   For each subject:
      Set the count to 0.
      For each time slot which is assigned a subject:
         Add 1 to the subject count.
      If a subject does not occur at least once and free slot(s) are present
         Find a free slot and assign the subject to it.

Fig. 5 – Pseudo Code for Second Stage of Repair Strategy for soft constraint 1

Soft constraint 2: There cannot be too many empty slots in between lessons.

[Analysis and Strategy]: For most classes, there will be more periods in a week than necessary, which will lead to free periods for classes. The free periods can also occur due to scheduling conflicts in the school timetable. To ensure that this constraint is not violated, free periods are moved after all the study periods.

D. Evaluation

The evaluation process calculates the cost of all the chromosomes in the new population. The cost function of the evaluator is as follows:

- Cost = Summation (Penalty)
- Penalties assigned:
  - Faculty crash = 5
  - Non-paired Labs = 10

The assignment of the penalties will actually filter out the chromosomes that violate the constraints.

E. Selection

After the evaluation process, every chromosome in the population is assigned with a cost. The selection operator will sort the chromosomes from the lowest to the highest cost. M chromosomes with the lowest cost will be selected for the next E.A generation.

F. Crossover

A single point crossover is operator is utilized in this implementation. The point of cross over is a randomly generated number. Crossover pairs are generated randomly for the entire population. All the gene(s) following the crossover site is exchanged between the pairs. This simple yet effective modification of the conventional crossover provides the necessary genetic variation.

G. Mutation

The mutation operator selects a chromosome randomly out of the new population, and chooses a gene randomly from the chromosome for mutation. The gene value of the chosen gene of the selected chromosome will be changed randomly.

V. CONCLUSION AND FUTURE WORK

It has been seen that “gaatt.e” performed directed evolution on a simplified Lecture Timetabling problem, and produced timetables void of hard constraint violations. These results were very promising.

The major improvement that could be made to its performance on the Simplified Lecture Timetabling Problem would be the further inclusion of Repair Strategies. Also, the fine tuning of its performance parameters could provide faster evolution.

The program was seen to be readily scalable to the Complete Lecture Timetabling Problem. This would be achieved by increasing the size of the University and by incorporating further constraints on the timetable.

GAs effectively demonstrated an ability to solve complex optimization problems. Notably, this served to provide a very thorough introduction to the techniques employed and incorporated by Genetic Algorithms.

REFERENCES


