Automated Timetable Generation using Genetic Algorithm

Mithila Shetye¹, Anusha Pai², Cassandra Dias³

¹P.G Student, Dept. of IT, P.C.C.E Verna Goa  
²,³Asst. Professor, Dept. of CE, P.C.C.E Verna Goa

Abstract— Timetable generation is a very burdensome and time consuming task. This is usually done ‘by hand’, taking several days or weeks of iterative repair. Timetable generation is the NP-hard problem, which is very difficult to solve using conventional methods. A highly constrained timetabling problem can also be solved by evolutionary techniques. We must determine an acceptable assignment of the time slots and rooms to the courses based on a variety of their requirements. This project will try to reduce the difficulties of generating timetable by using Genetic Algorithm. By using Genetic algorithm, the time required to generate time table will be reduced and the generated timetable will be more accurate, precise and free of human errors. Main goal is to minimize the number of conflicts in the timetable. System generates timetable for each class and faculty, in keeping with the availability calendar of teachers, availability and capacity of physical resources (such as classrooms, laboratories and number of computers) and rules applicable at different classes, semesters, faculty and subjects level. Project results will possibly show that the genetic algorithm improves the fitness of the population for each generation, and it returns a feasible solution, even for the most constrained benchmarks.

Index Terms— Genetic Algorithm, Automated timetable, timetable

I. INTRODUCTION

The class timetabling problem is a typical scheduling problem that appears to be a tedious job in every academic institute once or twice a year. In earlier days, time table scheduling was done manually with a single person or some group involved in task of scheduling it manually, which takes a lot of effort and time. Planning timetables is one of the most complex and fault-prone applications.

A timetable is a table that shows the time, room and faculty for a set of courses that are to be scheduled. Timetable must be designed to satisfy all user requirements or else it would be a waste of time and resources. To produce the semester timetable for your academic department, you would need to schedule courses in such a way that there would be a faculty, room, time and students for each course to be offered as well as satisfy some constraints.

There are basically two types of constraints, soft constraints and hard constraints. The system can generate a valid timetable even if soft constraints are violated. But if we violate the hard constraints then the timetable is no longer valid. In course timetabling the hard constraint will be that at a time faculty cannot take lecture in two different classrooms and after each lecture at least one lecture relaxation should be given to the faculty. An optimal and feasible solution would be obtained if and only if all the hard constraints are satisfied.

II. PROBLEM DESCRIPTION

Timetabling problem involves scheduling of courses, faculties, and rooms to a number of time-periods or timeslots in a week. This project aims to construct a timetable for ME course of PCCE
which has theory as well as practical sessions. For ME, the course is divided into four semesters, two semesters per year. They can be referred to as odd and even semesters.

A. Problem Definition

Participants consisting of the timetable are as given below:

a. faculty f1, f2, f3,……fa
b. studentgroup s1, s2, s3,……sb
c. classrooms/labs r1, r2, r3,……rc
d. course/subject c1, c2, c3……cd and
e. sessions s1, s2, s3,…se

A is the set of all faculties. This class has an ID and the name of the faculty. It also contains an object ‘availability’ which defines the availability of the faculty for the week. Class availability specifies the day and the time interval during which the faculty is available.

B is the set of students group on semester basis. This class has an ID and the name of the student group, as well as the number of students.

C is the set of classrooms and laboratories available in the college to conduct a lecture and practical respectively. This class has an ID and the name of the classroom, as well as a flag that differentiates between class and a laboratory.

D is the set of subjects. This class has an ID and the name of the course.

E is the set of sessions that holds a reference to the course to which the class belongs, a reference to the faculty who teaches, student group that attend the class, the duration of the class (in hours) and the classroom where the faculty teaches the group.

Classes are conducted on 5 days of the week i.e., from Monday to Friday. The duration of each timeslot is 1 hour. The number of sessions, faculties, courses, groups, classrooms and labs to be allotted to a slot depends on the input you provide.

With all these inputs provided, the system will generate separate timetable for the individual class automatically, in keeping with the availability of teachers, availability and capacity of physical resources and rules applicable at different classes, semesters, faculties and course level.

B. Constraints Involved

The constraints that we consider are classified as hard and soft. Hard constraints are those to which a time table has to hold fast in order to be satisfied.

Hard constraints involved are:

1) A classroom can be occupied by one course at a time.
2) Sessions can only be scheduled during normal working hours.
3) A faculty can only be in one place at a given time.
4) Time slots are assigned to theory sessions, practical sessions and tutorials if any.
5) A Group is assigned only one course at a given time.
6) A group should not have two sessions of same course in a day.
7) There should not be an empty slot between two sessions for a group.
8) Two continuous sessions should not be assigned for a faculty.
9) Time preferences for lecturers: Lecturers should be able to specify preferred day or time of teaching (morning/afternoon, evening)
Invading the above constraints will cause the time table to be unfeasible. Soft constraints for this constrained optimization problem are actually the preferences or choices student and teachers would like to take advantage of. They can be as follows.

1) Only 1 hard subject a day: Some subjects are known to be harder than others.
2) Room/Lab exclusion: It should be possible to exclude rooms/labs from the timetabling process, so they are not considered by the generation algorithm.
3) Faculty is assigned maximum of two theory sessions in a day or 1 lecture and 1 practical session in a day
4) Session can be compulsory or elective
5) No sessions on Friday for final year students in order to give them enough time to work on their project.

III. APPROACH

Among the several techniques, Genetic Algorithm is the most frequently used and preferred over the others. Genetic algorithm works like this:

A population is created with a group of individuals created randomly. The individuals in the population are then evaluated. The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task. Two individuals are then selected based on their fitness, the higher the fitness, higher the chance of being selected. These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly. This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.

![Fig.1 Structure of GA](image)

A. Initialization of Population

In order to get the feasible solution faster, instead of taking completely random samples in the beginning, we have taken initial chromosomes in such a way that they satisfy few hard constraints initially using the below functions.

seminsters, two semesters per year. They can be referred to as odd and even semesters.
sessionSerialise()
This function assigns an id to all the sessions in JsonDe to identify the sessions uniquely and hence avoid the occurrence of duplicate sessions.

prioritizeProfessor()
Here we find a list of faculties who can take lectures on specific day and time and give them a higher priority compared to other faculties.

SessionGenerator()
Here will identify all the sessions taught by professor that have higher priority than others and the same are added to global session list. These sessions will be assigned first in the timetable at random.

sessionPrioritize()
The sessions with maximum duration are identified, and added to session list. These are given high preference and are assigned next at random if these sessions were not present in global session list. After assigning them at random, they are also added in global list after assigning.
When we select a slot at random, we have below four functions:

slotChecker(): if there are enough slots for the session according to session’s duration.

getPreference(): if selected slot has any preference parameter.

labValidator(): if the allotted room is a lab in case of practical session

roomChecker(): if the assigned room is available

All the remaining sessions that are not present in the global session list are now assigned to remaining time slots at random. Here we get our first three samples as initial population.

B. Fitness
Now we need to assign a fitness value to the chromosome. As mentioned earlier, only hard requirements are used to calculate the fitness of a class schedule. This is how we do it:
Each class can have 0 to 7 points.
If faculty has no other class at that time, we increment its score.
If group has no other class at that time, we increment its score.
No continuous two lectures for a faculty, we increment its score.
No two lectures of same course in a day, increment the score
No gap between two lectures for a group, increment the score.
Faculty is assigned sessions according to his/her availability schedule, increment its score.
If a class breaks a rule at any time-space slot that it occupies, its score is not incremented for that rule.
The total score of a class schedule is the sum of points of all classes.

\[
\text{Fitness} = \frac{\text{Schedule score}}{\text{Max score}}
\]

Schedule_score= sum of scores of all classes
Max_score=number of classes * 7

C. Selection
Using the three combinations, we get total 6 samples in our first generation. We sort these 6 samples in the descending order. The top three are added to the ranked list and are used as samples for next
D. Crossover

We have used one point crossover. We randomly select a day and interval of time. The time slots between these intervals are swapped between two chromosomes to obtain two child chromosomes. Procedure is followed as shown below.

CrossOverProcessor()
{
    copy chromosome 1 in tempcrossover;
    weekDay = random.Next(JsonDe.days);
    from = random.Next(JsonDe.to - JsonDe.from - 1);
    to = random.Next(from, JsonDe.to - JsonDe.from);
}

It will select a random set of continuous slots from 2nd chromosome on a random day and call matchmaker function.

The Matchmaker() function is used to check all the possibilities while swapping the timeslots and if it results in true value then the crossover procedure proceeds otherwise a new set is chosen again.

Matchmaker()
{
    if sampleone not null and sampletwo not null
    If durations match continue, else break;
    if sampletwo null and sampleone not null
    If sampleOne duration ==1, continue
        Else break
    if sampleone null and sampletwo not null
    If sampleTwo duration ==1, continue
        Else break
    if both are empty then continue
}

If matchmaker function calls crossover function, it works as given below:

Crossover()
{
    If both are not null → swap the sessions
    If slots in tempcrossover are null, replace the empty slots with chromosomeTwo slots and delete the corresponding sessions

generation. This process continues until we get feasible solution or it exceeds the limit of number of generations.
If slots in chromosomes two are empty, then move the slots in chromosome one to empty slots, and insert the corresponding sessions in empty slots of tempcoprocessor. If both are empty, do nothing

}  

E. Mutation  

In mutation we exchange timeslot of one session with the timeslot of another session of the chromosome. We select a random time slot and identify the session assigned to the same. We then find the timeslot in which the selected session starts. We then replace this session with another session at random in the same chromosome. We achieve this using a function blankchecker that checks the feasibility of exchanging sessions.

Blankchecker will return the number of empty slots. If these slots are enough to fit the selected session then mutation takes place, or else another session is selected at random.  

IV. CONCLUSION  

As mentioned earlier, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. Here we are aiming to obtain a feasible solution with fitness value approaching one.

As we iterate through generations, from initial population towards the feasible solution, the fitness value begins to improve from 0.2 to 0.4.

As the highest ranking solution's fitness has reached a plateau and successive iterations no longer produce better results, we terminate the program. Further improvement in the fitness was not achieved because of numerous possible combinations of sessions and timeslots.

For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, Hill climbing and swarm intelligence and methods based on integer linear programming.

REFERENCES  


[2] Solving Timetable Problem by Genetic Algorithm and Heuristic Search Case Study: Universitas Pelita Harapan Timetable Samuel Lukas, Arnold Aribowo and Milyandreana Muchri Faculty of Computer Science, Universitas Pelita Harapan, Indonesia  

