Automated TimeTable Generation

Arijit Pramanik

December 31, 2016

Contents

1 Introduction

TimeTable generation has been a problem for institutes across the world. Though manual generation is within human capability, given resources are ample, you should try to go with the automated version as you save your computation time and have a sample space of possible optimized such timetables to choose from. The approach taken up is a go-back approach. You make mistakes and correct them as you repent for them. Same goes for your life too :).

1.1 Generation approach

APPROACH : Genetic Algorithms

The approach that we have taken takes in all the input and then it has all the requisite constraints that are pre-fed. So, the algorithm generates random timetables with some sort of prior repair strategies and then the constraints are deployed on them which gives them costs. You continue generating timetables so as to narrow down your sample space and get the optimal solution.
Note that you may have multiple optimal solutions but the algorithm gives you one as it stops as soon as you get one. Not satisfied.... Run it once again and keep doing it until you get your desired output which you think suits your college the best.

2 Description of Genetic Algorithms

We evolved from apes..... nope from *Archaebacteria*.

2.1 Actual Algorithm

So we generate three random numbers: a random day ranging from 0 to number of working days-1, a random slot from 0 to number of slots -1, a random room from 0 to number of rooms -1. So you have many subjects(say x) and a subject can have many classes scheduled (say y). So the random generation which takes place for the timetable schedules these xy classes or lectures in these slots(from 1 to xy).
Now you have generated these arbitary timetables. Then you breed them and consider it to be biparental (*As we considered here...* you can change it by changing the cross over rate variable in breed_colony function). Then you get children timetables and they get some of their mother timetable traits and some of their father timetable traits.
Then the child test_tube undergoes various tests. If it has less errors which are measured by costs and weights, then it becomes part of the population called solution colony. In this way, we take genrated timetables, calculate their costs, and maintain a population of three timetables which have the lowest costs among all the generated timetables. The solution colony now consists of three timetables as the population_size has been kept at three which has by far produced the fastest timetables.

2.2 Biological analogy

Human children are produced by the cross over of two parents. So it is biparental.. the same as our algorithm. Human population has an upper cap ....its fixed(*say*). So if the population is more than 9 billion, ISIS takes them away and kills them. Similarly, if we have more than three timetables in our solution_colony, then we kill off the costlier timetables and keep the lower cost ones. If the child born is imperfect, then he/she is killed, else if it is healthier and has better survuval abilities than its parent, it lives. The test_tube timetable created from two parent timetables is discarded if its cost is more than parent timetables. The population size is such that evolution is the fastest and **stable**. So the population size has a fixed optimal value. Mutation takes place as in modern days and hence timetables have randomly shuffled slots...so that who knows by a mere lucky shuffle you get the correct one in a single shot. Just as mutation can sometimes give you superhuman abilities like the X-Men, but may kill you through cancer..if its not favourable. So the timetable may get accepted into the population if mutation yields a lower cost or killed off and discarded if its cost is higher.

3 INSTRUCTIONS and INFO

You might wanna check inputformat.ods

3.1 The main function

- you have the change the name of the input file here from where the inputs are to be read.
- you can decide at which cost of the optimal timetable you would stop by manipulating the maximum allowed cost. Again, I suggest 0.
- initialise_constraints will initialise the input file.
- find_average_cost will again calculate that whether the population from which you got the timetablw was highly optimised or not. Its just a measure so no fuss regarding that.
- breed_colony will breed the timetables and generate the army of timetables having a population size of 3 which is the value we have used

3.2 breed_colony

The function will take randomly two timetables from three and label them as father and mother. The function will randomly take few slots from mother timetable and few slots from father timetable and calculate the cost of the child timetable. It compares the cost with the previous members of the population and inserts in in the appropriate place and discards the rest. The solution_colony has timetables sorted in the order of costs.
No need to manipulate any parameters here also.

3.3 The Three Musketeers : Repair Strategies

- the repair_strategy actually allocates classes to all slots for each and every room for all the classes for each subject. It may allocate them 0 or multiple times.
- the repair function deallocates those slots which are out of purview of the course working hours or if the particular subject has zero affinity towards the room. So bookings are just cancelled.
- the repair_strategy_0 actually ensures that each and every class is allocated precisely once. It checks for how many times classes have been allocated and hence takes prompt action to book the classes if not booked earlier.

3.4 initialise_colony

This function generates the random timetables. Initially, all the slots are available. This function makes use of repair_strategy_0 to book each and every class exactly once since bookings are all null or 0 times for all classes when timetable with all slots free is generated.

3.5 find_average_cost

You might want to know which population is better. All nations have its best swimmers: Phelps for USA and Sun Yang for China and others. But you might want to know whether as a bunch, USA swimmers are better than the chinese swimmers, so you need to evaluate their population traits. So this function is a measure whether the population has all optimised timetables or some are not. If the average cost is low, then the optimised timetable comes from a superior breed.
P.S. Of course, USA is much better.

3.6 calculate_cost

This calculates the cost of the individual timetables...
Dude, remember to assign weightages for each of the constraints.

3.6.1 Unavailability

For lecturer, room and obviously course, you supply the time_slots as and how you like but using hh:mm format with start and end time seperated by spaces. Enter however many time slots you want and corresponding matrices are generated automatically.

3.6.2 room_mismatch and room_affinity

Certain infra of room needs to be present. We have incorporated lab and projector so farand it schedules classes accordingly. For more infra, you can assign weightages to each of the components, create a room infra and decide which room is most suitable for each class in an order. Zero would mean strict denial for that room for a particular class.

3.6.3 related class clash

This gives the number of instances that the different classes of the asme subject are scheduled in the same slots in different rooms.

3.6.4 room too small and room_compat

room too small gives the number of such cases the strength of the class is more than the capacity of the room which they are allocated.
room_compat will not schedule a class of lower strength in a class with very high capacity if a room with a lower capacity is already available in that slot.

3.6.5 lecturer_double_booked

Gives us those cases where the same lecturer is given the same slot for different classes in different rooms....Basically can't take more than one lecture at a time

3.6.6 lecturer_overtime

This will check whether a lecturer is scheduled or alloted more number of **hours of lectures** than he is supposed to take in a week as well as for each day.

3.6.7 subjects_per_week

This makes sure and gives us those instances when a lecturer devotes lesser time to a subject than he should have done in a week(or the entire semester. You can change accordingly). Input is in **hours of lectures**.

3.6.8 class_continuity

This checks whether classes in a timetable can be scheduled by minimising the free available slots between them. It makes sure that students have classes in continuation..and do not have to wait longer for classes to finish because of no classes for them in between. This is implemented **course-wise**.

4 Database

InnoDB was the internal storage engine used on our PHP.

4.1 Logistics

- upload lecturer, classes, room data onto db.
- optimised timetable is uploaded onto db.
- create free slots will tell us the availability of slots to be used when the courses have been finished or not. The duration field for each of the classes will tell us its duration in the input file.
- Make changes in db for the various properties of classes : it may get over while the semester is in full swing, the lecturers might change on any arbit day of the week, etc.

4.2 Tasks

To add courses in the xth week

- firstly fetch the (x-1)th time table data
- Add the new courses in the free slots of the timetable taken care by the code. I might say that make the lecturer availability in the previously scheduled classes prior to new courses zero so that the algorithm can't even think of using those slots. Lecturer_unavailability is a huge must-be-followed constraint.
- Upload the updated timetable in the db

5 Code Optimisation

Intense profiling on Netbeans has been done and rest assured that the code is more than moderately optimised. You need to dig through enough to find out the creeks where the control flow wastes its time. You should aim for the functions listed inside breed_colony function.

6 Limitations

- We had achieved a peak speed which had been demonstrated, but following certain changes instructed to us, it was not possible to maintain the superb optimisation characteristic of a student of the glorius CSE dept. of IITB. But , however, profiling of the code will be submitted and done as much as possible so that someone might speed things up a bit.
- Still now, class_continuity and room_affinity are not working in full swing. They will fail in corner cases, but you won't come across such institutes too often. They often mess up the code and as constructed statistically, slows down code and may enter into a never ending loop.
- The dynamic database could not be presented. The current one just takes in the various data objects and displays correctly only for the first input. Only god knows what would happen if second inputs are provided. It is completely possible that you might get the correct desired timetable. Week-wise, but not day-wise changes could be incorporated.