Solving an Exam Scheduling Problem Using a Genetic Algorithm

Dave Kordalewski
davekordalewski@gmail.com

Caigu Liu
liucaigu@gmail.com

Kevin Salvesen
kevin.salvesen@gmail.com

Abstract

This concise, one-paragraph summary should describe the general thesis and conclusion of your paper. A reader should be able to learn the purpose of the paper and the reason for its importance from the abstract.

The Problem

The problem we investigate is a sort of scheduling problem. We are attempting to find the most satisfactory choice of when and where to hold exams, given a background of student course loads.

An instance of this scheduling problem consists of a number of days on which exams can be scheduled (d), the number of time slots in which an exam can be scheduled on any day (t), a set of rooms (R), a set of courses (C), and a set of students (S), each of whom (s) has a particular course load, some subset of the courses (L_s).

A schedule, then, is a mapping from courses to rooms at times, which we can express like this:

$$C \to \{(r,\ a,\ b) \mid r \in R,\ a \in \{1..d\},\ b \in \{1..t\}\}$$

The total number of different possible schedules in any problem instance is $|C|^{|R|dt}$ which is typically far too large for brute force search. For example, one instance that we examine (scheduling the fall semester exams at the University of Toronto in 2009) involves 603 courses, 7 days, 8 times, 43 rooms, and 21945 students. This instance admits approximately 10^{6695} different possible schedules.

Typically, with relatively loose constraints on the number of rooms and particular schedules of students, it is not difficult to find some consistent schedule; that is, one where no student is asked to write 2 exams simultaneously and no room has 2 exams occur in it simultaneously.

Rather than worry about hard constraints, we prefer a framework of soft constraints, where we try to find a schedule that makes the students and invigilators happiest overall, allowing the possibility that some student or room is left with an impossible exam schedule, which can, in the real world, be dealt with on an individual basis.

The timetable that a student $s \in S$ has under some particular schedule K may be represented in this way:

$$TT_s(K) = \{(a,b) \mid \exists r \in R, \exists c \in L_s, K(c) = (r,a,b)\}$$

which may, in general, be a multiset.

Similarly, the timetable for a room $r \in R$ is

$$TT_r(K) = \{(a,b) \mid \exists c \in C,\ K(c) = (r,a,b)\}$$

We define two quality functions, mapping schedules to real numbers in [0,1], one for students and one for rooms. These are meant to capture how much they "like" the schedule under consideration. (For instance, a student will rate poorly any schedule where she has 2 consecutive exams, and very poorly any schedule which expects her to take two exams at the same time.)

$$Q_s(K) = f\,(TT_s(K)) :\ \textit{Schedules} \to [0,\ 1]$$

$$Q_r(K) = g\,(TT_r(K)) :\ \textit{Schedules} \to [0,\ 1]$$

We will consider later how to define these functions in a reasonable way.

The quality of a schedule, then, is given by

$$Q(K) = \frac{\left(w_s\left(\sum_{s\in S}\frac{Q_s(K)}{|S|}\right) + w_r\left(\sum_{r\in R}\frac{Q_r(K)}{|R|}\right)\right)}{(w_s + w_r)}$$

where w_s and w_r are the weightings we can use to indicate that we care somewhat more about students preferences than rooms, giving a quality (or fitness, in the terminology of genetic algorithms) in the range [0,1] for any schedule.

Evaluating this function Q can be computationally expensive when $|S|$ and $|R|$ are large. Our work examines how to find a schedule with relatively high fitness considering that we will want to do this with as few evaluations of Q as possible.

We examine, first and foremost, a genetic algorithm approach to solving this problem, but also touch on a few other methods.

Approach and Implementation

Tools, algorithms, APIs, etc. that we used.

Evaluation

How we determined the success of our project. Did we solve it? Effectively? Labelled images, graphs and/or tables can be useful.

Conclusion

What have you learned because of your project? What can we learn from your project? Did it give you ideas as to things you might like to try next?