Genetic Algorithms Applied to University Exam Scheduling

Nicholas Jenkins, Tom Gedeon
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia
{nickj, tom}@cse.unsw.edu.au

Abstract

A method of scheduling complex examination timetables, such as found at most universities is presented that utilises genetic algorithms. This extends previous work [1,2] by splitting the problem into two sequential components, both of which are solved using genetic algorithms. Also, it can be seen that some subjects are likely to be harder to schedule correctly than others, so a method of linking the reward for the clash-free scheduling of these subjects to scheduling difficulty is described. Finally, in situations where a clash-free timetable cannot be found, it is very desirable to minimise the number of students who experience clashes.

Key Words: heuristic, genetic algorithms, scheduling, timetabling, examinations, combinatorial optimisation

Introduction

Scheduling examinations for significant student bodies is a real and recurring problem faced by many secondary and tertiary academic institutions. The simplest form of the problem is to allocate students, invigilators and examinations to rooms and times in such as way that no student is scheduled to be sitting more than one exam at any given time. Increasing enrolments worsens the timetabling problem, as do moves towards more modularized degrees [1], as both increase the density of the problem – the probability that any two given subjects will have some students in common. One approach that can be used in scheduling examinations is genetic algorithms.

Genetic Algorithms

Genetic Algorithms (GAs) are a class of heuristic search algorithms. Their natural analogy is the phenomenon of population genetics, whereby the Darwinian principal of survival of the fittest is applied to species over time. Some individuals are deemed to be fitter than others, because more of their offspring survive. This process involves offspring taking portions of DNA from both of the parents.

When transferring this analogy to a computer algorithm we can apply GAs to combinatorial optimisation problems that we wish to solve. Most crucially though, we must decide how we are going to encode a solution to that problem. This solution encoding is called a chromosome, and is analogous to DNA. This chromosome can be represented as a bit string of fixed or variable length. A model, called a fitness function, is then created that is intended to reward good attributes of a chromosome and punish bad attributes of a chromosome by adding to or deducting from the total fitness of that solution. If the fitness function is correctly designed, and a good representation has been chosen, then a GA will usually find good solutions given sufficient processor time, but with increasing problem size, finding a workable solution in a reasonable time can become an issue.

Timetabling Exams

Timetabling exams is the process of assigning exams to rooms and timeslots. It is a multi-constrained combinatorial optimisation problem that is known to be NP-hard [3]. In addition to the basic constraints of the timetabling problem (eg. no clashes, room capacity never exceeded), extra constraints of varying importance can be added dependant upon the institution in question. One example is that more popular subjects be timetabled earlier to allow the markers more time to assess the larger number of papers.

As the GA is ultimately intended to be applied to determining an end of session examination timetable at the University of New South Wales, Sydney, Australia, some assumptions from this environment were made. It is assumed that there are two examination periods per day (morning and afternoon), that multiple exams of different length can be assigned to the same room, and that no exam will take longer than one examination period (which in practice means a maximum of three hours).

Previous work in this area include studies by Corne, Fang and Mellish [1] at the University of Edinburgh, UK and by Burke, Elliman and Weare [2] at the University of Nottingham, UK. Corne *et al.* created a system that is used for scheduling times for the Master of Science exams at the University of Edinburgh, but did not apply the GA to scheduling rooms. Burke *et al.* used a direct representation that specified the time and location of an exam, but only considered exam spacing issues for adjacent exams.

Method

GA Engine and Platform Used

For this implementation, no special recombination or crossover techniques were used, meaning that the core GA engine is abstracted from the problem and can be replaced with another engine if desired at a later point. Currently a commercial GA engine is used called Evolver ™ [4], from AXCÉLIS Corporation. The platform used was an Intel ™ Pentium ® 100 MHz PC running Windows NT ®.

Data Used

The primary data set used to test the performance of the GA was from the enrolment data for the Ecole Des Haute Etudes Commerciales, Montreal, in the summer of 1992, which has the following characteristics: 2823 students, 81 subjects, and a problem density of 42% [5]. The data consists of a file containing the subject enrolments for each student on a separate line. When applying GAs to exam scheduling, real data should be used wherever possible above generated data, as the two can have significantly different characteristics.

Fitness function Overview

Disliking the idea of forcing [6] or backtracking [7], whereby illegal solutions are made legal by making small modifications, we have instead implemented a system whereby chromosomes that break hard constraints are not discarded, they are just made very unfit. This allows the good portions of overall bad solutions to survive if the good portion is very good, and the hard constraints are not excessively violated. This speeds up the evolutionary process, and ensures that the GA produces reasonable quality solutions within a few generations.

In the fitness function being used, qualitatively better solutions are defined as having a lower fitness value. Undesirable features of a solution are penalised by adding a penalty to the solution's fitness. A perfect solution is defined as having a fitness of zero, but due to the conflicting nature of some of the constraints, no solution will ever have zero fitness.

Representation

One of the most crucial steps in applying a GA to a particular problem is deciding how the solution to that problem is going to be represented. A wasteful representation will reduce the speed at which a solution is found and reduce the effectiveness of crossover between individuals in a population. Fortunately, an effective and intuitive representation for the exam-scheduling problem exists.

Initially a direct chromosome was used based on the representations used in [2] and [1]. For instance, for four exams an 8-tuple was used, where {2,5,4,1,3,7,3,6} would be translated as "exam 1 at time 2 in room 5, exam 2 at time 4 in room 1", and so forth.

2 Step GA

Currently a modified version of the above representation is used. To improve the quality of results produced, the scheduling problem has been split into two sequential steps – this means that instead of one large problem where the GA has try to determine the optimal solution from scratch, each stage focuses on improving different aspects of the solution.

The first sub-problem is the determination of acceptable concurrent groupings of exams and assigning of exams to rooms. The chromosome used here for m exams was a chromosome of size 2m, where for each exam the room and time *group* is considered. This step primarily considers the hard constraints of the problem – that is the constraints that are of paramount importance. These are that the total number of clashes is minimised (preferably to zero) and that instances of exceeding the number of seats available in the given set of rooms are minimised (again preferably to zero).

Other soft constraints are also considered, but are weighted to be much less significant. These include reducing the amount of wasted capacity in each room, and minimising the total room rental cost for the exam period. This step is left to run until the best solution has not improved for over a thousand generations, at which stage it will terminate.

The second step in the GA is a translation from the conceptual grouping of exam to actual exam timeslots. For the assignment of groups of exams to actual timeslots in an exam period with n timeslots, a chromosome of length n was used. Note that to retain the conceptual grouping established in the first step, each time group must map uniquely to an actual timeslot. This step is only concerned with soft constraints, specifically how the arrangement of exams can best be improved to satisfy the users of the system – namely the students and the exam markers. To this end, there are two primary soft constraints in this step – arranging exams so that the larger exams are placed earlier in the timetable to give markers with more papers a greater a marking time, and arranging exams so as to improve the exam spacing for students. For exam spacing, a timeslot is inserted for every overnight gap (to reflect that an exam one afternoon followed by an exam the next morning is generally considered preferable to two exams in one day), and four timeslots are inserted for every weekend gap. Adjacent exams are then defined to have a high penalty, exams with one gap between have a moderate penalty, and so forth. Exams separated by 2 ½ days spacing or more have no penalty. By altering the timeslots that exams are assigned to, the spacing for some individuals is likely to worsen, but on average it should improve.

The results of this two step process is a timetable tries to meet the hard constraints of the problem as best as possible (given a limited number of slots and rooms) and also tries to make the spacing of exams as painless as possible for the users of the timetable.

Conflict Matrix

A conflict matrix for an exam period of m exams is an m by m array that shows for any given two subjects whether of not they have students in common (and hence whether placing them in the same time group will create a clash). The idea of combining a conflict matrix with a GA is introduced in [2] – "if the value at row *i* and column *j* of the matrix is one then the exams *i* and *j* conflict, ie. have a student in common, otherwise the value will be zero." The conflict matrix used in the GA system we have implemented extends this idea by listing the number of students in common, not just a Boolean value and this has a number of benefits. As soft constraints are being used, it means that a penalty value dependent on the number of students who experience the clash can be used, which is advantageous when clashes are inevitable, as it rewards schedules which minimise the numbers of students who experience clashes. It also helps when considering exam spacing issues. [2] only considers adjacent exams, but by using the number of students in common in the conflict matrix, it is possible to calculate how good the exam spacing is over any number of gaps, not just adjacent exams (we ignored gaps over 2 ½ days between exams). For example, in a simple timetable with 3 sequential exams (1,2,3), the exam spacing penalty is: (number students in common between exam 1 & 2) * penalty for adjacent exams + (number students in common between exam 1 & 3) * penalty for exams with one gap between + (number students in common between exam 2 & 3) * penalty for adjacent exams.

Once this modified conflict matrix has been formed the raw student data is no longer needed and can be discarded from memory. This represents a significant memory usage improvement with large numbers of students. Furthermore, operations on the conflict matrix are significantly faster than searching through all of the students.

Difficulty Reward

It was found experimentally that the subjects with fewer clashes were being scheduled sooner than exams with a larger number of clashes – this is not surprising, but it does have the effect that when running the GA the total number of students who experience clashes falls very rapidly initially, and then decays very slowly. This gives the impression that minimal progress is taking place, and can eventually lead to a situation where only a handful of exams remain, but they conflict with exams in all the possible timeslots – this amounts to being trapped in a local maxima. This situation can be overcome by rewarding the GA more for scheduling "harder" exams. In this implementation the difficulty of scheduling an exam was determined from the conflict matrix - for each subject, the number of times a non zero entry occurred in that subject's row in the conflict matrix, a counter was incremented for that subject. Then over each other subject that this subject clashes with, we sum the value of their counters multiplied together to give a difficulty measure. Dividing difficulty measure by the average difficulty measure and multiplying by the number of exams gives as a percentage the difficulty of the total problem that this subject accounts for.

By creating a constraint based on the total percentage difficulty remaining, it was possible to have a higher reward for the scheduling of harder subjects, and to also get a better idea of how the GA was actually progressing. This constraint was used in conjunction with the conventional calculation of the number of students who experience clashes, and was weighted equally, so that in situations where clash-free exam timetables cannot be determined, the number of students who actually experience those clashes is minimised.

Results

Finding a clash-free schedule using the GA for the Ecole Des Hautes Etudes Commerciales, Montreal data set took less than eight hours for 22 timeslots. A clash-free timetable could not be found using less timeslots than this.

Discussion

Speed Performance

Given that it took 8 hours to produce clash-free results with a real data set, it is reasonable to ask if this is unacceptably long time. However, whilst it is always desirable to produce results quickly, it must be remembered that this GA will only need to generate 2 timetables per year, and that there is a sizeable time period between when subject enrolments can no longer be altered and when the exam timetable is announced. With this in mind, a GA that is needs to be left overnight to produce reasonable results should be acceptable.

There is an area of greater concern - the data set used had 81 exams, and 2,823 students, whilst the University of New South Wales data is likely to contain approximately 900 exams and 20,000 students. How the GA performance will scale with this approximately 10-fold increase in data size remains to be seen.

Quality of Results

Often exam scheduling algorithms are tested with randomly generated data sets, which has the disadvantages of not duplicating the patterns that can occur within real data, and of meaning that there is no independent way of comparing results between timetabling techniques. However, the data set used was from a publicly available archive [5], and had been used previously by Carter *et al.* [8]. Carter *et al.* used a number of variations on the graph colouring heuristic algorithm with numerous data sets, including this one, and were able to produce timetables in 17 or 18 timeslots for this data set. This compares very favourably to the minimum of 22 timeslots produced by this GA. However, a comparison has yet to be made as regards the efficiency of room allocation, as no data was available on the exam room capacities at the different institutions. Also the implementation of exam spacing differ – Carter *et al.*'s exam spacing tries to meet constraints specified by the instituion, which this GA simply tries to space the exams to give the reasonable gap between exams for most students. To date, no scheduling results from other GAs on these public data sets have been found, so no comparison of this implementation against other GAs has yet been made.

Future Work

There are several problems that remain to be tackled before this GA can be applied to the exam-scheduling problem at the University of New South Wales. For instance, the GA outlined may need to be extended to include split exams (where the students taking an exam are allocated to more than one room, often on the basis of surname or student number) which has to be used if the number of students in an subject exceeds the capacity of the largest room. How the time taken to produce clash-free results will scale problem size remains to be seen.

References

[1] D. CORNE, H-L. FANG and C. MELLISH (1993), Solving the modular exam scheduling problem with genetic algorithms. *Proceedings of the Sixth International Conference of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems,* Gordon and Breach Science Publishers, Langhorne, Pa., Chung, Lovegrove, Ali (eds.), pp 370-373.

[2] E. K BURKE, D.G. ELLIMAN and R.WEARE (1994), A Genetic Algorithm Based University Timetabling system, *Proceedings of the Second East-West International Conference on Computer Technologies in Education* (Crimea, Ukraine, 19th-23rd Sept), vol 1, pp 35-40

[3] S.EVEN, A.ITAI and A.SHAMIR (1976), On the complexity of timetable and multicommodity flow problems, *SIAM Journal of Computing*, **5**(4), 691-703

[4] EVOLVER ™ , Version 3.01 and 3.5 software, 1995 AXCÉLIS Corporation.

[5] Data used available from: ftp://ie.utoronto.ca/mwc/testprob/all_data.tar.Z

[6] R. NAKANO and T. YAMADA (1991), Conventional genetic algorithm for job shop problems, *Proc. 4th Int. Conf. On Genetic Algorithms*, UCLA.

[7] S. KENNEDY (1993), Five ways to a smarter genetic algorithm. *AI Expert* **8**(DECEMBER), 35-38.

[8] M. CARTER, G. LAPORTE and S. LEE (1996), Examination Timetabling: Algorithmic Strategies and Applications, *Journal of the Operational Research Society,* **47**, 373-383