Solving the Modular Exam Scheduling Problem with Genetic Algorithms

Dave Corne, Hsiao-Lan Fang
and Chris Mellish

DAI Research Paper No. 622

ABSTRACT

Scheduling exam timetables for large modular courses is a complex problem which often has to be solved in university departments. This is usually done 'by hand', taking several days or weeks of iterative repair after feedback from students complaining that the timetable is unfair to them in some way. We describe an effective solution to this problem involving the use of an appropriately configured genetic algorithm (GA). Using real student data from a large multi-departmental modular degree scheme, the method we describe never failed to find a significantly better timetable than those that were actually employed (produced by hand), always taking less than half an hour.

INTRODUCTION

In a general scheduling problem, events must be arranged around a set of timeslots, so as to satisfy a number of hard constraints and optimise a set of objectives. Types of scheduling problem differ in terms of the kinds of constraints and objectives involved. In this paper we examine what we call the MESP (Modular Exam Scheduling Problem). This typically arises in universities running large modular degree schemes, in which each student takes an individual selection of exams from a wide inter-departmental pool of modules, many outwith their own department. The events are exams, the timeslots are possible start-times for those exams, the hard constraints are that no student should take more than one exam at a time, and the objectives are to generally minimise pressure on students, so that as few as possible have multiple exams in a day, consecutive exams, and so on.

Typical MESPs are NP-hard, and strewn with local minima which make it particularly difficult to address by way of heuristic search or hill-climbing techniques. MESP complexity is also illustrated by the size of the solution space. Eg: if there are t possible start times, and e exams, then there are t^e candidate schedules. In the particular MESP which occurs within the EDAI[1], this was 28^{44} in 1992, or c. 5×10^{63}.
When an MESP is tackled, typically in a university or college department, but very similar problems often occur in industry it is usually addressed by hand (eg: by a course organiser). This involves producing an initial draft timetable, followed by perhaps weeks of redrafting after student feedback complaining about the latest draft. The initial draft is often based on merging different departments' teaching/course timetables; but in large modular degree schemes, the fact that many students from one department typically take courses in others, and the fact that there are different lecture timetables for different terms, makes this a recipe for finding local minima, which then typically need extensive repair if better solutions are to be found (and hence better solutions are often not found).

(GAs) provide a way of addressing hard search and optimisation problems. GAs are particularly good at finding global optima in very hilly spaces; for these reasons, we investigated the use of GAs on the MESP. Following a very basic description of GAs

[1] University of Edinburgh Department of Artificial Intelligence.

for the uninitiated (for an excellent introduction, see [1]), we outline our GA approach to the MESP. Finally, we outline experiments in which different GA variants were used on a typical, real MESP, describe the promising results that ensued, and discuss future work and implications.

GENETIC ALGORITHMS

If we want to maximise a function $f(x_1, x_2, ..., x_n)$, where each x_i can take on any of its own range of values from a set v_i, then we can do this with a GA as follows. First, randomly generate a population of P candidate solutions. Eg: if each x_i has range $v_i = \{0,1\}$, then this simply corresponds to generating P random n-bit binary strings. Each candidate solution is called a chromosome (or genome). Call this initial population the current generation, then, until the number of generations g has reached a specified figure, or until all the chromosomes in the current generation have converged (have the same fitness), do:

1. Evaluate (ie: apply f to) each chromosome in the current generation. Let the sum of all the resulting fitness scores be S.
2. Stochastically select $P/2$ pairs of chromosomes from the current generation to act as parents for the next generation, such that the probability of a particular chromosome c being selected is $f(c)/S$.
3. For each parent pair, apply a *recombination* operator, with probability p_R, which yields two child chromosomes from the parents. Also, to each child, apply a *mutation* operator with probability p_M. In this way a new population of P chromosomes will be produced. Call this the current generation, and return to step 1.

GAs vary considerably in the different choices for steps 2 and 3. Eg, the selection method ('roulette wheel' selection) we describe in step 2 above is common, but just one of a number of possible methods. In step 3, the general idea is that the children produced by recombination will tend to have higher fitness scores than the parents. This can easily be seen to happen if two highly fit parents are fit for different reasons, eg: parent pqr is fit mainly because of its third gene being an r, while parent xyz may be highly fit because of its first gene being an x. A recombination operator may then produce xqr, which combines good aspects of the parents to produce an even fitter child.

One typical recombination operator is one-point crossover. If we have two chromosomes, $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_n$, then a random number i is chosen between 1 and $n-1$; this serves as a crossover point. One child is then $x_1, x_2, .., x_i, y_{i+1}, y_{i+2}, ..., y_n$, while the other is $y_1, y_2, .., y_i, x_{i+1}, x_{i+2}, ..., x_n$. A more general crossover operation is *uniform* crossover, in which for i from 1 to n, the ith bit of child1 is randomly chosen from $\{x_i, y_i\}$, and the ith bit of child2 is x_i if y_i was chosen for child1, or y_i if x_i was chosen for child1. In *fixed-point* uniform crossover (`fpu`), a fixed number m of bit-positions are chosen from a parent; one child then has x_i in the ith position if i is one of the chosen positions, and y_i otherwise, and *vice versa* for the other child. In the experiments described later using `fpu`, m was chosen to be half the chromosome length.

Many other choices of recombination operator are possible, with different operators working best for different problems. Recombination, however, is the essential aspect of a GA which seems to give it enormous power in searching the fitnes landscape. On the other hand, because of the nature of recombination, there may be parts of the search space which will be unavailable without the presence of a *mutation* operator. Mutation acts by randomly changing the values in bit positions, perhaps (re)introducing a possibly useful value. Recombination operators are normally applied to a pair of parents with a probability p_R, where typically $0.5 < p_R < 1$, while mutation is applied with a probability p_M, where typically $0 < p_M < 0.05$.

APPLYING GA TO THE MESP

In applying a GA to a problem, we must specify both a representation and an evaluation function for candidate solutions. Here we describe these aspects with regard to applying GAs to the MESP, while the next section outlines experiments involving the use of GA variants on a real MESP.

Representation: The representation that we have successfully used is simply a list of numbers of length e (the number of exams to be scheduled), each element of which is a number between 1 and t (the number of timeslots available). The interpretation of such a chromosome is that if the nth number in the list is t, then exam n is scheduled to occur at time t. For example, the chromosome [2,4,7,3,7] represents a candidate solution in which exam 1 takes place at time 2, exam 2 takes place at time 4, and so on. Other work, on applying GAs to timetabling problems in schools (see [2]) has used an alternative representation where the *position* in a chromosome represents the timeslot, while what appears at that position is a set of exams. This however leads to the *label replacement* problem: crossover often may produce children which are not valid solutions in that some exams may not be scheduled at all, or scheduled more than once; this necessitates the use of a label replacement algorithm after crossover, to replace missing exams or remove duplicates. Our representation completely avoids this problem: crossover may certainly lead to missing or duplicated *timeslots* (so that at certain times no exams, or multiple exams, are scheduled), but these are perfectly valid, and possibly good timetables.

Evaluation: For a general MESP, the evaluation function must take a chromosome, along with the *student data*, and return a *punishment* value for that chromosome. The student data is simply a collection of lists, where each list is the set of exams to be taken by a particular student. Chromosome fitness can then be taken as the inverse of punishment (or 1/(1+punishment) — to avoid division by zero). The evaluation function may comprise a weighted set of individual functions, each 'punishing' the chromosome in terms of a particular punishable 'offence'. In the MESP we experimented with, the components of the evaluation function are functions which respectively count the number of instances of the following 'offences':

- A student taking more than one exam at once (weight = 30).
- ... more than two exams in one day (weight = 10).
- ... two exams in consecutive timeslots on the same day (weight = 3).
- ... an exam just before and another just after lunch on the same day (weight = 1).

In other MESPs, different sets of component functions and weightings (the above were chosen intuitively) may be more appropriate, Eg: components which treat very early exams, or perhaps occurences of four exams in two days, as separate offences. In general, the fitness function for an MESP, where c is a chromosome, d is the student data, $\{m_1,...,m_n\}$ is a set of functions which each record the number of instances of a particular 'offence', and $\{w_1, ...,w_n\}$ are the weights attached to these offences, is: $f(c,d) = 1/1 + sum_1^n w_i m_i^{c,d}$.

Evaluation is generally the computational bottleneck of a GA. With the MESP, and 'offence-counting' modules of the type we have described, it is easy to see that the time to evaluate a chromosome increases linearly with the number of students, and can involve many computations depending on the kinds of punishment being looked for. It so happens (which is partly the message of this paper), that the GA technique is so powerful that a typical MESP can be quickly and effectively solved despite this bottleneck.

EXPERIMENTS AND RESULTS

Our experiments used real data from the MESPs of the EDAI postgraduate AI/CS exams for 1991 and 1992, involving 60 and 93 students respectively (from two departments), and 38 and 44 module exams respectively (including modules from four departments) to be scheduled , each student generally taking a selection of 8 exams from this pool. The different GA variants below were applied to both problems:
- **GA1**: basic GA with p_R at 0.7, p_M at 0.003.
- **GA2**: Inverse Square Pressure; as GA1, except that instead of using the inverse of punishment as the fitness function, we use the inverse square.
- **GA3**: GA2 + Elitism: here, one instance of the best chromosome in a generation was always copied into the next generation (note: because of the stochastic nature of selection in a GA without elitism, there is no guarantee that the best chromosome in one generation will appear in the next).
- **GA4**: GA3 + Operator Rate Interpolation (ORI); ORI involves gradually decreasing p_R and increasing p_M with each new generation.

We report on four experiments, each using one of the above GA variants on both the 1991 and 1992 timetabling problems. In the GA1 and GA4 experiments, we also experimented with different crossover operators.

SUMMARY OF RESULTS

The results we describe represent averages over 10 trial runs of 300 generations, with a population size of 50 in each generation. GA1 always managed to produce timetables comparable in fitness to the actual timetables used within 300 generations when `fpu`[2] was used, and GAs 2, 3 and 4 performed at least as well as the actual timetable whatever crossover was used, but `fpu` crossover was always best. In terms of the fitness score of the best chromosome after 300 generations, GA2 was 25% better than GA1, GA3 was about 250% better than GA1, and GA4 was about 400% better.

Using GA4 with `fpu` on the 1991 problem reliably resulted in a timetable better than the actual one, and the best (of ten trial runs) was a timetable with zero punishment – ie: no instances of consecutive exams, or more than two exams in a day (and certainly no clashes!) for any student. The actual timetable used for these exams had a punishment of 37, involving 11 consecutive exams offences, and 4 'before-&-after-lunch' offences.

In the 1992 case, again ten out of ten trials of GA4 with `fpu` crossover produced better timetables than that produced by the course organisers. The best from ten trials was a timetable with a punishment of 3 (only one occurence of a student having consecutive exams). The actual timetable used for these exams had a punishment of 101, involving 33 consecutive exams offences, 16 before-&-after-lunch offences, and 2 three-exams-in-a-day offences.

CONCLUSIONS

We have described the MESP, a common and very hard problem which occurs frequently in schools and universities. We have found that a simple, traditional GA using one-point crossover can quickly produce results comparable to human schedulers on a typical MESP. Further, combining elitism, operator rate interpolation, inverse square pressure, and `fpu` crossover leads to reliable production of optimal or near-optimal timetables, faring much better than an unaided human team. Our approach is straightforward to implement, and hence should be easy to adopt in any department or institution which continually needs to solve MESPs. The work already done is described fully in [3].

There is, however, a great deal more work to be done. So far, we have shown that GAs show great promise on timetabling problems of the kind and size reported in this paper. Also, at the time of writing the system has just been used for the 1992/93 version of this problem, involving yet more exams and more students, resulting in the production of a successful timetable. We have no comparative results for this particular latest application; this is to be expected however, because the course organisers who now use the system certainly do *not* want to go to the trouble of producing an exam

[2]The fixed number of positions swapped each time being half the chromosome length (ie: half the number of exams involved).

timetable by hand, now that they don't need to.

Nevertheless, we intend to pursue systematic studies from which we can learn how our approach scales up to larger problems, how the approach compares with conventional timetabling methods, and how performance varies with parametric, representational and algorithmic variations in the GA configuration.

Conventional computer-based timetabling methods concern themselves more with simply finding the shortest timetable that satisfies all the constraints, usually using a graph-coluring algorithm (finding sets of exams that can be scheduled at the same time corresponds to finding sets of vertices in a graph which are not adjacent), and less with optimising over a collection of soft constraints. As this is an NP-hard problem, conventional methods use approximate graph-colouring algorithms which usually find a reasonable, if not optimal, solution, though may perform arbitrarily badly. No such system that we know of, however, attempts to optimise over constraints of varying importance such as exam-consecutivity, etc ... ; this is because it is hard to fit priority based optimisiation into the conventional method. The GA approach, on the other hand, can deal with hard and soft constraints in a uniform way; further, adding or changing the importance of a constraint simply corresponds to adding/altering a component of the fitness function, rather than extensively revising the algorithm itself. At the moment, then, our approach is incommensurable with the conventional approach in that it attempts to solve a harder, more constrained problem. It still remains to be seen, however, how well our approach scales up to larger problems, both on its own and in comparison with conventional methods.

Another important avenue to explore is the more general problem, usually handled by conventional methods, in which exams (or lectures) may occur in a set of rooms with different capacities. Investigating this requires us to modify the chromosome representation and evaluation function, and will very shortly be tried via a project at the EDAI; the method used in [2] handles this more general problem, but suffers from a fairly poor choice of representation.

Examining performance sensitivity will come by testing a large number of GA variants on a standard corpus of problems, coupled with some theoretical work. The first few techniques of a long list that we intend to examine in this respect are: performance *vs* variation in population size; performance *vs* altered interpretation of representation (ie: we can rewrite the evaluation function to interpret a chromsome as an implicit representation of a timetable which satisfies all the hard constraints; this slows down evaluation, but may potentially improve overall speed and solution quality); performace *vs* different penalty settings for constraint violations, and so on.

Finally, we hope to have described how it is possible to greatly ease the burden on course organisers, and also ease the pressure on students at exam time, by applying a GA to the modular exam scheduling problem. We are unsure about the precise limitations of the technique at present, but feel confident about its general wider applicability. Work continuing at the EDAI will hopefully soon produce more conclusive performance data. We also feel confident that the general GA approach to timetabling will apply well to some other important problems of a similar form. An appropriate example is the problem of timetabling paper presentations at conferences with parallel sessions; if each delegate was given the opportunity to provide the organisers with their individual preferences regarding the papers to be presented, a GA based timetabling system very similar to the one we describe could arrange for parallel sessions to be organised such that delegates are collectively satisfied as far as possible, in terms of minimising the degree to which two presentations a delegate wishes to see are scheduled to occur at the same time. We plan to do this, with the help of the organisers, for the next European Conference on Artificial Intelligence.

REFERENCES

1. Goldberg, D.E., Genetic Algorithms in Search, Optimisation & Machine Learning, Addison Wesley, Reading, 1989.

2. Abramson & Abela "A Parallel Genetic Algorithm for Solving the School Timetabling Problem", Technical Report, Division of I.T., C.S.I.R.O, April 1991.

3. Fang, H-L., "Investigating Genetic Algorithms for Scheduling", MSc Dissertation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK, 1992.