A Genetic Algorithm Based University Timetabling System

Edmund Burke, David Elliman and Rupert Weare
Department of Computer Science,
University of Nottingham,
University Park, Nottingham, NG7 2RD.
e-mail: ekb@cs.nott.ac.uk

Abstract

The annual construction of an Exam Timetable is a common problem for all institutions of higher education. Quite often it is done by hand or with the limited help of a simple administration system and usually involves taking the previous year's timetable and modifying it so it will work for the new year. Many British institutions are now introducing the concept of the modular degree. This gives the students much greater flexibility in what courses they take as well as giving a much greater choice. For the timetabler, this, and the recent growth in student numbers, means that the timetable will be more constrained than ever. It is no longer good enough to use the previous year's timetable. Every year a new timetable must be produced to take account of staff, student and course changes causing a necessarily large amount of administrative work.

We present a prototype system for University Timetabling of both exams and courses based on the use of Genetic Algorithms. This will include an interactive windows based iconic user interface so that the user can work with the system and home in on a good solution.

1. Introduction

Timetable scheduling is the problem of assigning courses or exams to periods and to rooms. There are two types of University schedule: the course timetable and the exam timetable. These are related to each other but can be quite different. For example, generally, more than one exam will be held in each exam hall at any particular time whereas it would be extremely unlikely that any institution would allow two courses to take place in the same room. Also, the exam halls are shared between all departments within the institution as opposed to each department using its own rooms. This means that, practically, the exam scheduling process must be carried out centrally by the university. For the purposes of this paper we will consider exams rather than courses. The methods described, however, would be equally applicable to course timetabling.

The process of finding a period for each exam so that no two conflict has been shown to be equivalent to assigning colours to vertices in a graph so that adjacent vertices always have different colours [Welsh + Powell 67]. This in turn has been proved to lie in the set of NP-complete problems [Karp 72] which means that carrying out an exhaustive search for the timetable is not possible in a reasonable time. Many heuristic exam timetabling algorithms have been proposed, mostly based on methods used to colour graphs. These can often produce good timetables but usually ignore the possibility that there may not be sufficient seats for all the exams assigned for a particular period and do not allow for any search of the timetable solution space to take place.

2. Genetic Algorithms

Genetic Algorithms are powerful general purpose optimisation tools which model the principles of evolution [Davis L. 91]. They are often capable of finding globally optimal solutions even in the most complex of search spaces. They operate on a population of coded solutions which are selected according to their quality then used as the basis for a new generation of solutions found by combining (crossover) or altering (mutating) current individuals. Traditionally, the search mechanism has been domain independent, that is to say the crossover and mutation operators have no knowledge of what a good solution would be. However, it has been shown [Bruns93][Burke et al.94] that by using domain dependent operators good, if not better, results may be achieved.

In our case, a solution is a timetable which we can consider to be an assignment of exams to periods and to rooms. A Genetic Algorithm starts by generating a set (population) of timetables randomly. These are then evaluated according to some sort of criteria. An example would be how many times any student has to sit two exams

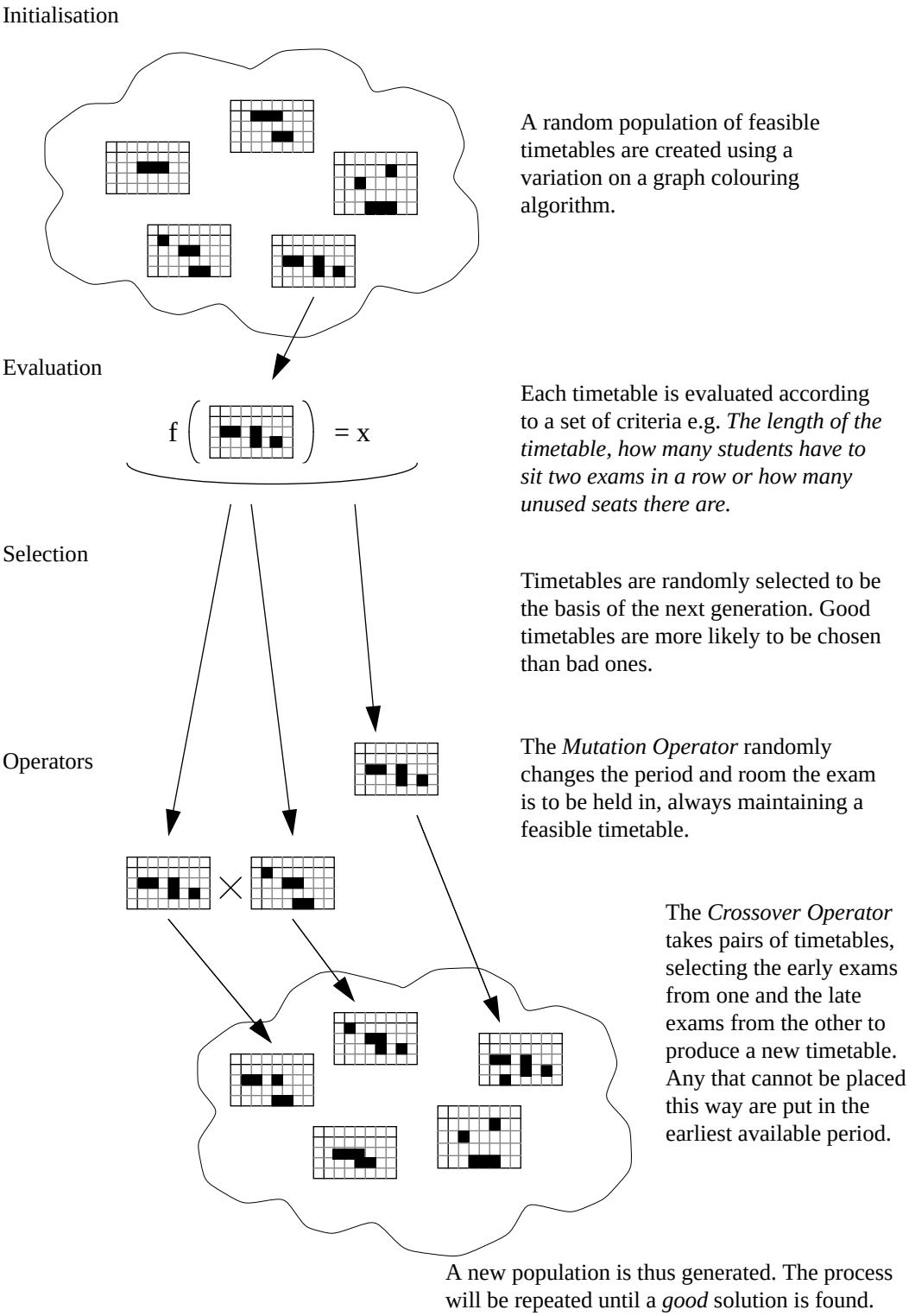

Figure 1: The Genetic Algorithm Process

in a row. On the basis of this evaluation *population members* (timetables) are chosen as parents for the next generation of timetables. By weighting the selection process in favour of the better timetables, the worse timetables are eliminated while at the same time the search is directed towards the most promising areas of the search space (see figure 1).

The crossover operator works by taking two *population members* and combining them somehow to produce one or two offspring. Traditionally this is done by randomly selecting a point (gene) in the coded solution then

appending the part of the second solution after that point to solution one up to that point and vice versa. The mutation operator is only applied to one solution at a time and involves the random variation of one particular gene. This adds a limited random element into the search and may reintroduce potentially useful *genetic material* that has been lost earlier in the search.

3. Timetable Generation

Two fundamental constraints govern the production of a timetable. These are that no student or invigilator can be in more than one place at a time and that there must be sufficient seats to house all the students present. We will call a timetable that satisfies these constraints a *feasible* timetable.

Just because a timetable is feasible, unfortunately, does not mean it is good enough to be used. Many other criteria exist which may be used to judge the quality of a timetable. The most common of these is that a student should not be expected to sit two exams in adjacent examination periods. A particular institution may also wish that only exams of a similar length are scheduled at the same time in the same room or that larger exams come first to allow more time for them to be marked. In the end, the only real way to judge whether a timetable is a good one or not is if the institution will use it. In this paper we will describe a prototype genetic algorithm based timetabling system which will allow feasible timetables to be optimised as the particular institution sees fit through the use of an appropriate graphical user interface.

Genetic Algorithms have been successfully used to schedule exams in a number of cases. Corne et al. use a fairly traditional approach where each gene represents the time at which its particular exam takes place with crossover and mutation operators as described above. This system, with a number of time saving improvements, is in current use at the University of Edinburgh. Paechter takes a different approach where the gene for each exam not only specifies when it is to be taken but also how to search for a new period if, after crossover, the exam is causing a conflict. If the exam may not be placed in any of the periods then it is left as unscheduled unlike the previous system which allows infeasible timetables.

The approach we use incorporates domain specific knowledge to ensure that none of the candidate solutions are infeasible. In particular, a direct representation is used that includes not only when, but also where an exam is to be taken. This will allow the timetabler to stop the algorithm running at any point and find a choice of *good* feasible solutions. Also, instead of using a fixed number of periods and having either conflicting or unplaced exams, the length of the timetable in periods is left to vary. Clearly an exam schedule that is longer than the allowed number of periods is undesirable but with the changeable nature of the problem we cannot predict how long the "optimal" timetable would be and it is possible that a feasible timetable with the required number of periods may not exist. The timetabler may affect the length of the timetable by weighting a length based penalty value in the evaluation function, a high weighting for shorter timetables and a low weighting for longer ones.

Heuristic based crossover and mutation operators are applied that use specific domain knowledge so that the two fundamental constraints are not violated. They also improve the timetable itself. The selection operator used is standard for Genetic Algorithms as it is independent of the representation.

Various fitness (quality) functions may be used and adapted to produce *tailor-made* solutions for a particular institution. This allows us to claim a much greater generality than has been possible in previous scheduling methodologies which are very often only usable with very specific requirements. In the case where non-feasible timetables are also allowed, there is often the danger that when other desirable but not necessary criteria are introduced, they may swamp the fitness function so no usable timetables are produced.

4. Some Example Results From The Genetic Algorithm

The operators briefly described above have been implemented to replace the standard operators in the GENESIS system [Grefenstette 90]. Test problems with one hundred exams to be scheduled in four rooms with capacities 40, 80 and two of 160 respectively, were generated. For each one a conflict matrix is generated. If the value at row i and column j of the matrix is one then exams i and j conflict, i.e. have a student in common, otherwise the value will be zero. The probability that any two particular exams conflict is given by p which in our example takes the values 0.2, 0.4 and 0.5. The sizes of each exam were also generated randomly to be values between 1 and 100. A powerful heuristic assignment algorithm, described fully in [Burke et al. 93a], is also used for the purposes of comparison.

The evaluation function used was:

$$\begin{aligned} & ((\text{ LENGTH OF TIMETABLE IN PERIODS }) - 10)^2 \times 50 \\ + \; & (\text{ NUMBER OF ADJACENT CONFLICTING EXAMS}) \times ADJ \\ + \; & (\textstyle\sum_{\text{periods}} \sum_{\text{used rooms}} (\text{ SPARE SEATS })^2) / 1000 \end{aligned}$$

where *ADJ* (the penalty value for two exams that conflict being in adjacent periods, known as a second order conflict) is a constant and SPARE SEATS are only counted when there is at least one exam in the room. The value of *ADJ* was varied to experiment with the length of timetable produced. If *ADJ* is given a high value i.e. adjacent exams are considered to be very bad then clearly the resulting timetable length is going to be long. If a smaller timetable is required then the value of *ADJ* must be reduced.

In the following table results are presented for values of *p* between 0.2 and 0.5 and values of *ADJ* between 1 and 100. In each cell, the value for a timetable is given as 'the number of adjacent exams/length of the timetable (measured in periods)' of the best results after 10 trial runs of 300 generations, each with a population size of 100. The first underlined value is the best timetable in the initial population.

	ADJ = 1	*ADJ* = 20	*ADJ* = 50	*ADJ* = 100	Heuristic
p = 0.2	<u>177/13</u> 162/13	<u>162/14</u> 119/14 101/15	<u>159/14</u> 106/14 102/15 101/16	<u>143/14</u> 110/15 89/16	141/13
p = 0.4	<u>295/13</u> 288/13	<u>273/17</u> 217/17 188/18	<u>243/18</u> 188/18 173/19 159/20	<u>218/20</u> 179/18 171/19 163/20 143/21	284/15
p = 0.5	<u>268/19</u> 264/18 251/19	<u>247/20</u> 219/20 204/21	<u>246/20</u> 167/24 155/26 141/28	<u>235/23</u> 147/26 137/28 126/32	249/20

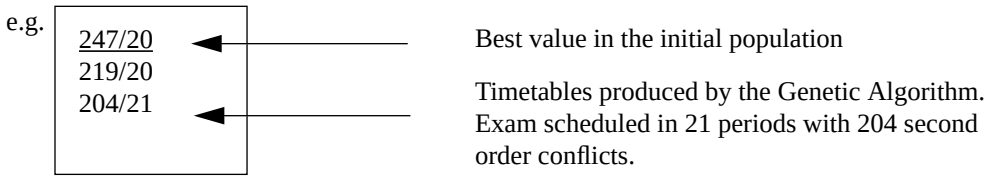

Figure 2: Results of the Genetic Algorithm

In every tested case, the genetic algorithm improves, often significantly, on the initial best timetable and except when *p* = 0.2 on the heuristically generated solution. This is because when the probability of conflict is low, the most constraining factor is the accommodation constraint for which the heuristic solution employs a limited search. As should be expected, the longest timetables are produced when *p* and *ADJ* are given high values.

5. Using the System

As we have described, the Genetic Algorithm will aim to maximise the evaluation function it has been given as has been described. This may be just one criterion or perhaps the weighted combination of several as in the above example. In reality, however, the timetabler, when working manually, would be simultaneously trying to balance many different aims and objectives. Some of these may not even be easily expressible in the evaluation function. The evaluation function will return a single value whereas, in fact, there will actually be a surface or frontier of

solutions all of which return the same fitness value. In this case, the timetabler must find some other means of choosing the one to use.

As has been said before, the only real test of whether a timetable is of high quality is whether the institution will use it. Clearly, the more control over the search process that the timetabler has and the wider the scope of the system, the more likely it is to be accepted. Using a Graphical User Interface (G.U.I.) will allow the user to oversee the workings of the algorithm and contribute where necessary to the scheduling process. It also has the advantage that in the inevitable situation when something completely beyond the imagination of the timetable system designers occurs, it can be trivially solved.

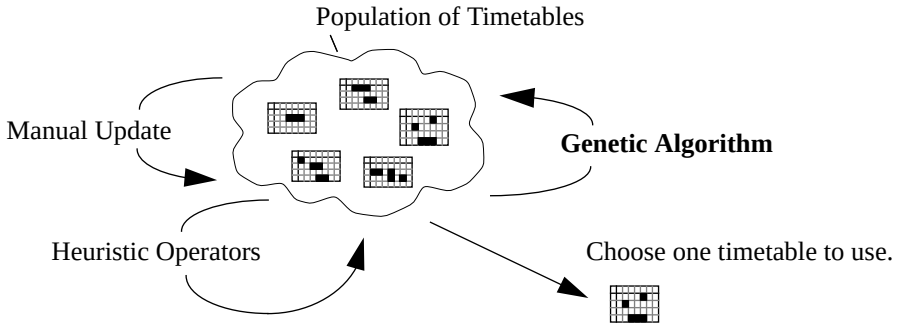

Figure 3: Searching for a Timetable

In using the system, the timetabler may use his or her own knowledge to aid the search as well as used other heuristic operators which act on the timetables to maximise various criteria while the Genetic Algorithm still provides the basis of the search and gives a choice of possible solutions. The algorithms may be left to run or the interface used to guide the timetabler to the area of greatest conflict.

The interface being developed is based around a spreadsheet idea. Timetables may be displayed with time on one axis and either student, staff, room or indeed anything else involved in the process on the other, for example, if a department has only one slide projector then it may be useful to include this in the system so it cannot be double booked. This allows database type queries to be performed. For example, “What is Dr.Johnson’s timetable?” or “When is lecture room 3 being used for French?” These may be displayed next to each other. Alternatively, the more standard timetable layout with days on one axis and times on the other may be used. By using icons for each exam, the user may drag one across the screen from one time period and drop it in another thus rescheduling it, or move a room icon and drop it on an exam to signal that that exam is to be held in that room. Double clicking on any particular icon will reveal greater information about it, for example, within an icon representing a room, information may be stored as to which building it is in, how big it is and which department it is normally associated with.

Figure 4 shows one of the possible timetabling modes with the standard timetable layout. Largely based on the original pin boards, the interface has two feeder areas which may be used as a temporary store for exams. The right hand one of these is used to hold exams that are about to be placed in the timetable by the user while the other stores exams currently unscheduled.

Figure 4: The Timetabling Interface

6. Conclusion

We have described a general purpose prototype University scheduling system which is capable of:

- Handling many different forms of timetabling constraint while only ever dealing with feasible timetables.
- Generating high-quality solutions despite the increasing intractability which has resulted from modularisation.
- Providing a choice of several different good schedules from which the user may choose the best.
- Directing the timetabler to the most constrained parts of the timetable so that, if necessary, adjustments may be made manually.
- Allowing database queries to produce a schedule for any staff member, student, room or item of equipment.
- Generating a personalised view of the timetable for each member of staff, communicating this over the campus network, and inviting on-line comments on perceived quality.

References

Bruns R. (1993) “Knowledge-Augmented Genetic Algorithm for Production Scheduling”, IJCAI ‘93 Workshop on Knowledge based Production Planning, Scheduling and Control.

Burke E.K., Elliman D.G. and Weare R.F. (1994) “A Genetic Algorithm for University Timetabling”, AISB Workshop on Evolutionary Computing, Leeds.

Burke E.K., Elliman D.G. and Weare R.F. (1993a) “A University Timetabling System Based on Graph Colouring and Constraint Manipulation”, *To appear in the Journal of Research on Computing in Education.* Vol. 26. issue 4

Burke E.K., Elliman D.G. and Weare R.F. (1993b) “Automated Scheduling of University Exams”, *Proceedings of I.E.E. Colloquium on “Resource Scheduling for Large Scale Planning Systems”*, Digest No. 1993/144

Carter M.W. (1986) “A Survey of Practical Applications of Examination Timetabling Algorithms” *OR Practice* 34, 193-202.

Corne, D., Fang H-L., Mellish, C. (1993) “Solving the Module Exam Scheduling Problem with Genetic Algorithms.” *Proceedings of the Sixth International Conference in Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Chung, Lovegrove and Ali (eds.), 1993, pages 370--373.

Davis L. (1991) “Handbook of Genetic Algorithms” Van Nostrand Reinhold

Fang, H.L. (1992) “Investigating GAs for scheduling”, MSc Dissertation, University of Edinburgh Dept. of Artificial Intelligence, Edinburgh, UK.

Grefenstette J.J. (1990) “A User’s Guide to GENESIS version 5.0”

Karp R.M. (1972) “Reducibility among Combinatorial Problems” In *Complexity of Computer Computations*, Plenum Press, New York.

Paechter B., Cumming A., Luchian H. and Petriuc M. (1994) “Two Solutions to the General Timetable Problem Using Evolutionary Methods” *To appear in IEEE WCCI*

Welsh D.J.A. and Powell M.B. (1967) “An Upper bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems” *Comp. Jrnl.* 10, 85-86.