The ideas presented represent the basis for a prototype system for University Timetabling of both exams and courses. This will include an interactive spreadsheet-like user interface so that the user can work with the system and home in on a reasonable solution. By a reasonable solution we mean one that the user is satisfied with, rather than a fully optimal one.

The final system should be capable of:

(1) Generating high-quality solutions despite the increasing intractability which has resulted from modularisation.

(2) 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.

(3) Inducing rules from staff comments which can be stored in the knowledge base and then used in objective functions to improve the quality of current and future schedules.

References

[1] Burke E.K. Elliman D.G. and Weare R.F. (1993) “Extensions to a University Exam Timetabling System”, IJCAI ‘93 Workshop on Knowledge-Based Production Planning, Scheduling and Control.

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

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

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

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

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

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

[8] 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.

table 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.

Table 1: Results of the Genetic Algorithm

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

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 possibly 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.

4. Conclusion

As has been mentioned, the quality of a timetable is important. It has often been said that the only measure of a good timetable is whether an institution will use it. By the very nature of the genetic search, the timetabler will be presented with a set of solutions that may be judged independently on their merits. The results show that using this method, timetables may be produced to suit individual requirements. There will necessarily always be a trade-off between the number of adjacent exams and the length of the timetable. By varying the value of *ADJ*, the timetabler may minimise the number of adjacent exams for a given length of exam period. Clearly, the inclusion of other constraints into the process is very simple.

We intend to experiment with further, more powerful, assignment heuristics, such as the one described in [2] which should produce even better solutions than the ones presented above. It is also intended to develop other advanced genetic operators, that will maintain a valid timetable, and different conflict graph structures such as may be expected in a large university.

advanced crossover works as follows:

```
Select two parent timetables.
Pick two crossover points at random: $x,y$.
  Take the exam assignments 1..$x$-1 and $y..n$ from parent 1
                    (where $n$ is the number of exams)
  For each exam $x$ to $y-1$ Place the exam according to the following priorities:
    • Same Period, same room.
    • Same Period, different room (biggest possible).
    • Earliest possible period and room.
```

The crossover operator acts by firstly trying to keep the exams where they are, i.e. directly inherited from the second parent, then by trying a different room, so maintaining the number of adjacent conflicting exams and if those fail, by putting the exam in the earliest possible period and room. This has the effect of keeping the timetable length small. By always trying the biggest rooms first it tends to try to keep the exams to large exam halls rather than having them scheduled in many different rooms.

Advanced Mutation

The new mutation operator works in much the same manner, essentially taking an exam at random and moving to a new place in the timetable. This can alter both period and room assignments so the operator can affect all information represented in a timetable.

```
Select one timetable.
Select one exam at random.
  Choose a new period for the exam.
  Place the exam according to the following priorities:
    • New Period, biggest possible room.
    • Earliest possible period (and room) after the chosen period
```

3. Results

The operators described have been implemented to replace the standard operators in the GENESIS system [7]. 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 according to probability p that given two different exams, they conflict i.e. have a student in common. The sizes of each exam were also generated randomly to be values between 1 and 100. A powerful heuristic assignment algorithm, described fully in [2], is also used for the purposes of comparison.

The evaluation function used was:

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

where *ADJ* (the penalty value for two exams that conflict being in adjacent periods) 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 time-

2.2. Initialisation

The initialisation algorithm uses a simple random sequential graph colouring method which is outlined below:

```
For each population member:
    Generate a random ordering of exams
    Taking each exam in turn:
        Find the first period in which the exam may be placed without conflict
        Try and find a room for it (biggest possible)
        If it does not fit in any room then try again with the next appropriate period
```

This can quickly produce large populations of random feasible exam schedules.

2.3. Evaluation Function

The evaluation function can clearly be made up of any timetabling related factors so for instance, if it were hoped that larger exams appeared earlier on for marking purposes then it would be possible to include that in the function. In this case, we concentrate on three particular common requirements:

- The length of the timetable.
- The number of second order conflicts
  (These occur when a student must take two exams in a row).
- Spare capacity in each of the rooms.

No penalty is needed for constraints (1) and (2) of section 1 because the initialisation algorithm and new genetic operators will ensure that they will never occur.

These requirements will produce an evaluation function of the form:

$$\begin{aligned} & f(\text{ LENGTH OF TIMETABLE IN PERIODS }) \\ + \; & g(\text{ NUMBER OF ADJACENT CONFLICTING EXAMS}) \\ + \; & h_1(\Sigma_{\text{periods}} \Sigma_{\text{rooms}} h_2(\text{ SPARE SEATS })) \end{aligned}$$

where f, g, h_1 and h_2 are all standard functions that may be altered to affect the weighting of each particular requirement. Each institution may alter these values in accordance with its own particular rules or indeed it may include other possibilities.

2.4. Genetic Operators

Advanced genetic operators are introduced because normal domain independent operators will hardly ever produce a feasible schedule. Both operators utilise simple heuristic ideas similar to those used to produce the initial population to guarantee that every offspring is also feasible.

The selection operator need not be changed as it works only with the fitness functions and is independent of representation.

Advanced Crossover

As well as ensuring that each offspring is feasible, knowledge augmented operators may use non-payoff information to guide the GA towards good solutions. They must also allow the new individuals to inherit part of the structure from each of their parent timetables. The as separate phases. First exams are split into separate non-conflicting sets possibly by a graph colouring algorithm [8], or indeed by a genetic algorithm [6], and then each period set, taken in turn, is assigned to rooms. Generally this has worked in the past on the grounds that there has been plenty of space in the University exam halls to house as many as may be required. In most higher education institutions that situation is now changing with student numbers being much larger (making (2) more difficult to satisfy) and with the exam period being squeezed because of modularisation (making (1) harder to satisfy also). Small changes to an existing timetable can propagate throughout the entire timetable. If one exam has to be rescheduled, (because of room constraints, say) then the whole timetable may need extensive repair before a new acceptable version can be found.

The approach we use specifically addresses this problem by using a direct representation scheme that includes not only when, but also where, an exam is to be taken. Domain specific knowledge is also used in the form of two very simple heuristic assignment algorithms, one for periods and one for rooms which are used to create advanced domain dependent crossover and mutation operators. These are designed to take advantage of all the information represented by each individual timetable allowing the search algorithm to cover the full range of feasible solutions. There is no need to have an extra penalty value in the evaluation function because infeasible solutions are never permitted.

2. A Genetic Algorithm

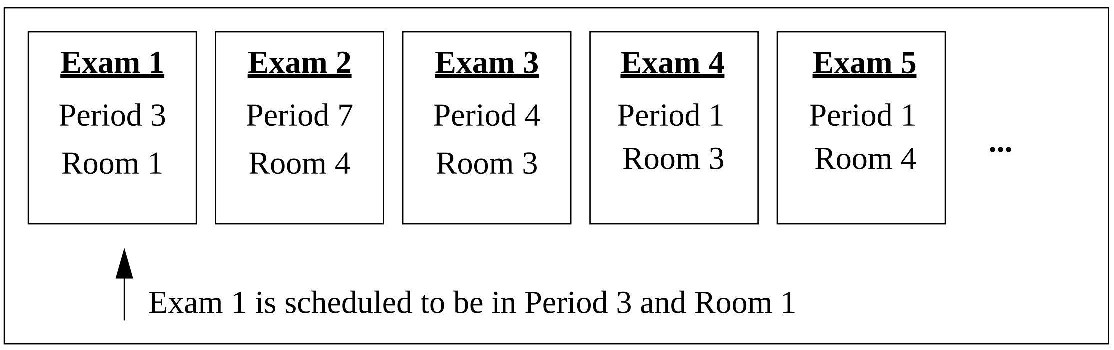
An advanced genetic algorithm capable of producing good exam timetables is presented below. It searches in the solution-space made up only of feasible timetables using knowledge augmented operators to generate new timetables that are also feasible. It can search through the entire search space because the finishing date of the exam period is left unspecified. Although most institutions have a fixed examination period this is necessary to allow a complete search. It also has the added advantage that, if possible, the period can be shortened, giving extra time for marking of papers, a common requirement in many institutions. This may be included in the evaluation function.

An initialisation procedure is described as the starting population needs also to be made up of feasible timetables.

2.1. Problem Representation

The representation chosen is very similar to that in [6] except that a value for rooms is also specified. The example below shows the first five genes (exams) and their particular assignments. For each exam, the period and room are specified, and coded as the following non-negative integer:

$$\text{PERIOD} \times \text{NUMBER OF ROOMS} + \text{ROOM NUMBER}$$



A Genetic Algorithm for University Timetabling

Edmund Burke, David Elliman and Rupert Weare
Department of Computer Science,
University of Nottingham,
University Park, Nottingham, NG7 2RD.

Abstract

The introduction of modularity into institutions of higher education and increasing student numbers mean that timetables are becoming more complex and difficult to schedule. A modular course structure leads to the possibility of large changes in the numbers and requirements of students taking courses from year to year. Every year a new timetable must be produced to take account of staff, student and course changes. This often involves a large amount of administrative work.

In this paper, we describe a genetic algorithm capable of solving such timetabling problems which, because it only operates on already feasible schedules, can be easily incorporated into an interactive spreadsheet-like user interface so that the user can work with the system to produce a high quality timetable.

1. Introduction

Exam scheduling is the process of assigning exam entities, each of which represents a set of invigilators, a set of students and a set of conditions under which the exam must take place to a particular slot in the timetable and a particular room. There are many different methods to create exam timetables, mostly heuristic assignment algorithms, and several computer timetabling and administration systems exist but quite often they can only cope with very particular situations. Each timetabling problem is as individual as the institution from which it originates. To create a generalised system, it is necessary to take into account as many different requirements and to allow the introduction of as many different timetable rules and constraints as possible.

There are two fundamental constraints that are universal to all timetabling problems and that no feasible timetable may violate. These are:

(1) No student must be in more than one place at once (no conflicting exams)

(2) There must be sufficient seats to house all the students present (no overfull rooms).

Most other constraints may be considered desirable but not essential, it is these that the timetabler will want to optimise but only within the framework of feasible schedules.

The quality of the timetable is important. Clearly, no candidate timetable may violate these two fundamental constraints, even slight violations cannot be allowed. Other constraints, for example, no-one is to have exams in two adjacent periods, may be highly desirable but may be ignored, if necessary, to produce a workable timetable. Constraints like these are amenable to the use of optimisation techniques, like genetic algorithms.

Conventionally, the two assignments of exams, to periods and to rooms, have been considered