A Memetic Algorithm for University Exam Timetabling

Edmund K. Burke
James P. Newall
Rupert F. Weare

Department of Computer Science
University of Nottingham
University Park
Nottingham NG7 2RD
UK

Contact: Edmund Burke <*ekb@cs.nott.ac.uk*>
Keywords: Exam timetabling, memetic, evolutionary, local search

ABSTRACT

The scheduling of exams in institutions of higher education is known to be a highly constrained problem. The advent of modularity in many institutions in the UK has resulted in a significant increase in its complexity, imposing even more difficulties on university administrators who must find a solution, often without any computer aid.

Of the many methods that have been applied to solving the problem automatically, evolutionary techniques have shown much promise due to their general purpose optimisation capabilities. However, it has also been found that hybrid evolutionary methods can yield even better results. In this paper we present such a hybrid approach in the form of an evolutionary algorithm that incorporates local search methods (known as a *memetic* algorithm).

1. Introduction

1.1. The Timetabling Problem

Many constraints involved in exam scheduling vary from institution to institution. However it is generally accepted that the following two constraints are fundamental to any timetabling problem:

- No entity must be demanded to be at more than one place at a time. In exam timetabling this would mean that no student can sit more than one exam at any one time.
- For each period in the timetable, the resource demands made by the events scheduled for that period must not exceed the resources available. In exam timetabling it is important not to schedule more exam sittings in a room than there are desks.

These two rules define a *feasible* timetable, and were the objective merely to find a feasible timetable, the problem could be solved using a variety of methods such as heuristic assignment[1] . However, we are usually interested in finding *good* feasible timetables, which can be defined as one that is practical and with which the user is happy. It follows that many different institutions will have differing views on what constitutes a good timetable[2] and therefore the engine of any automated timetabling system must be capable of satisfying the wide range of constraints that may be specified. Common constraints that may be demanded by users include:

- No students should have to take two exams in adjacent periods.
- No student should have to take two exams on the same day.
- Exam A must be scheduled before exam B
- Exam A must be scheduled at the same time as exam B (as they contain same or similar material).
- An exam must be conducted in a particular room (as it requires special resources only available in that room).

It is unlikely that, given the length of the average exam period and the number of suitable rooms available for exam sittings, that all of the above constraints could be satisfied completely. The optimisation capabilities of Genetic Algorithms [3] have been found to work well on various timetabling problems[4–6] , usually by first ensuring satisfaction of the conditions for a feasible timetable, and then optimising the number of desirable but not essential constraints that are satisfied.

1.2. Genetic Algorithms

Genetic algorithms are a general purpose optimisation tool based on Darwin's theory of evolution. They have the capability to produce optimised solutions even when the dimensions of the problem increase and for this reason they have been successfully applied to a wide variety of problems[7] .

Genetic Algorithms operate on a population of solutions represented by some coding. Each member of the population consists of a number of *genes*, each of which is a unit of information. New solutions are obtained by combining genes from different population members (*crossover*) to produce offspring or by altering existing members of the population (*mutation*). For our purposes we will also define light mutation as any small alteration and heavy mutation as any large scale alteration. A simulation of 'natural selection' then takes place by first evaluating the quality of each solution and then selecting the fittest ones to survive to the next generation.

1.3. Memetic Algorithms

The concept of a *memetic algorithm* was first introduced by Moscato and Norman[8] to describe evolutionary algorithms in which local search is used to a large extent. This idea has further been formalised by Radcliffe & Surrey[9] and a comparison between memetic and genetic algorithms made. A *meme* can be thought of as a unit of information that reproduces itself as people exchange ideas. A meme differs from a gene in that as it passed between individuals, each individual adapts the meme as it sees best whereas genes are passed unaltered.

The main advantage gained from the use of memetic algorithms is that the space of possible solutions is reduced to the subspace of local optima. To illustrate this, consider a single variable function *f(x)* described by the curve in figure 1. The child initially produced by applying a crossover operator is outside the space of local optima, but by then using the information available about the height of the curve immediately to either side we can navigate an upward path until the highest local peak is found.

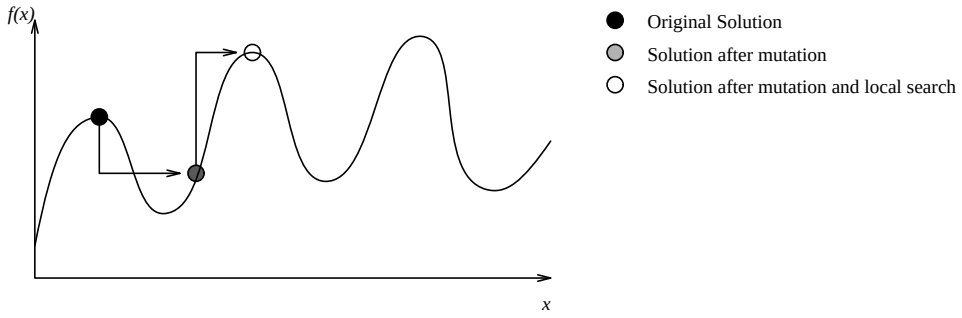

Figure 1 - An Example Memetic Operation

The addition of local search to the normal genetic operators inevitably has some computational expense but this can be justified by the reduction in search space that must be explored in order to find the optimum solution.

2. The Memetic Timetabling Algorithm

The algorithm presented below deals exclusively with fixed length timetables. This approach was chosen because, for most timetabling problems an upper limit on the number of periods is specified which should not be exceeded under normal circumstances. If at any time an exam cannot be scheduled in any of the available periods it is placed in a list of unscheduled events. This has the effect that the population may contain timetables that are feasible but incomplete. However given sufficient time and an adequate number of periods these should evolve into complete good quality timetables. The main techniques used in the algorithm are a combination of both light and heavy mutation followed by hill-climbing.

2.1. Solution Representation

Each solution in the population is represented as a number of memes. Each meme contains information on what exams are scheduled in which rooms for a particular period. A further meme is used to hold exams which could not be scheduled in the prescribed periods. Figure 2 shows an example of an encoded solution, where e_i is exam number i.

Period 1				Period n		Unscheduled
Room 1	e1, e2, e4			Room 1	e5, e22, e24	e53, e49
Room 2	e3, e7			Room 2	e37, e57	
Room 3	e11, e36			Room 3	e45	
Room 4	e90, e27			Room 4	e63, e52	

Figure 2 - the Problem Encoding

2.2. Initial Population Generation

The initial population is generated using a weighted roulette wheel to choose which period to place each exam (based on the exams already placed). The routine to generate a member of the initial population can be summarised as:

1. Remove an event *e* at random from the list of unscheduled exams. If all exams have been scheduled then finish.
2. For each period in the timetable, determine if it is legal to place *e* in that period. If so, calculate a measure of how 'good' it would be to do so using the function:

$$\frac{numCommon+Size+1}{penalty+1}$$

where *numCommon* is the number of neighbours that *e* has in common with the exams already scheduled in that period, *Size* is the number of exams already scheduled in the period and *penalty* is the cost (as used in the evaluation function) of placing *e* in that period. This heuristic measure is intended to, firstly, schedule events that have similar sets of neighbours together. This is a graph colouring heuristic that is often, but not always, a characteristic of a good timetable. Secondly, it tries to minimise the penalty caused by scheduling *e* by biasing the roulette wheel to those periods which can accommodate *e* with lesser penalties.

3. Construct and execute a roulette wheel based on the values calculated to choose a period in which to place *e*.

This mix of random and heuristic assignment was chosen in order to produce a higher quality initial population than the normal purely random generation while still keeping the degree of diversity that is desirable in Genetic Algorithms generally.

2.3. The Evolutionary Operators

There has been some debate as to whether crossover should be the main operator in a Genetic Algorithm. It has also been suggested that maybe local search has a part to play[8, 9] , this approach being known as “memetics”. A directed mutation approach has also been found to achieve good results[5].

The algorithm presented here has been found to work well (see section 3) using a combination of light and heavy random mutation, followed directly by application of hill climbing techniques. Figure 3 shows the cycle of evolution for the algorithm.

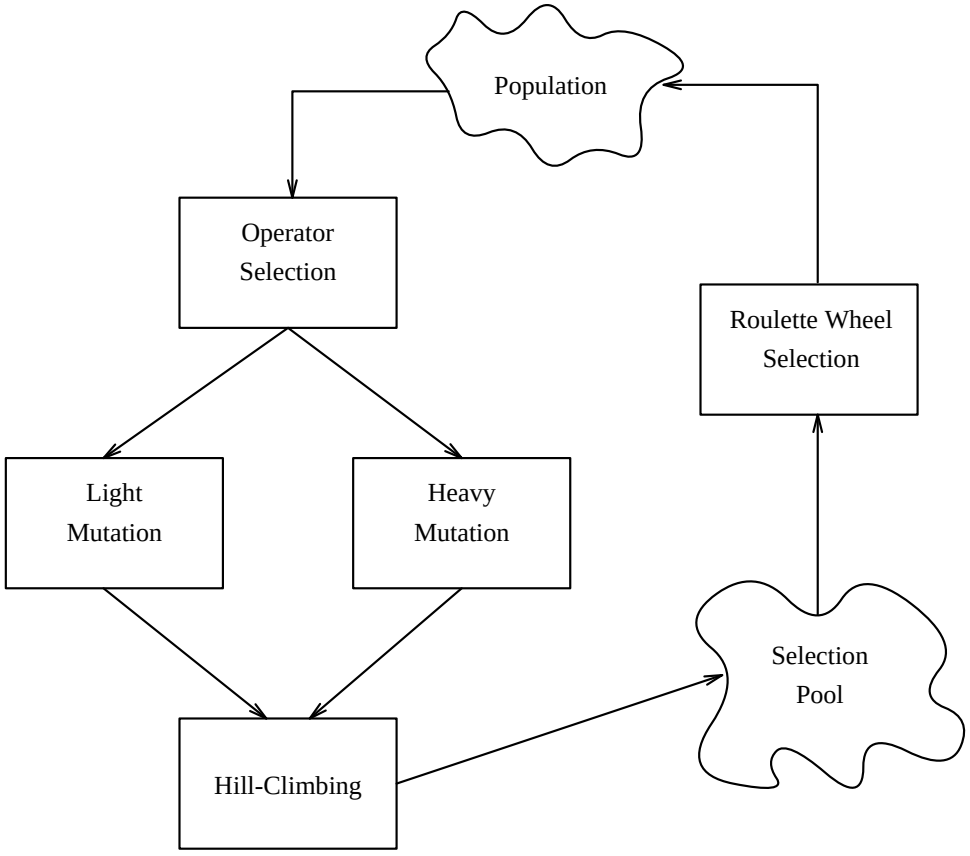

Figure 3 - The Evolutionary Operators

2.3.1. The Light Mutation Operator

A random mutation operator is used to perform light mutation on population members. This chooses a number of events at random from any point in the timetable and reschedules them at some other legal period. This is then directly followed by an application of the hill-climbing algorithm shown below.

2.3.2. The Heavy Mutation Operator

In order to find good solutions quickly a heavy mutation operator is used. This operator disrupts one or more whole periods in a timetable. This preserves well constructed periods in a timetable while randomly rescheduling exams in the remaining periods in order to find new higher quality solutions. The procedure for determining whether or not a period is to be preserved can be summarised as:

- Take each period *i* in the timetable in turn.
  - Calculate the *penalty* arising from the events in period *i* assuming the events scheduled in period *j* (where $j > i$) will remain fixed.
  - If this penalty is lower than the *average penalty* in the population then give the period a probability of being disrupted of

$$P(disrupt) = \frac{penalty + 0.2}{2.average}$$

  - Else,
    - If the last period was not disrupted then do not disrupt this period but instead disrupt the next period.
    - Otherwise disrupt this period.

Once it has been decided which periods will be disrupted the exams contained within these periods are grouped together. Each exam in this group is then removed at random and rescheduled in the first legal period. This operation alone rarely produces any substantial improvement but when followed by an application of the hill-climbing algorithm to reach the new local optimum substantial improvements can be made.

2.3.3. The Hill-Climbing Operator

As Hill-Climbing techniques generally require a great deal of evaluations, this operator uses a technique of calculating the penalty incurred by scheduling an event in a particular period, given that the other events are fixed. This has the advantage that when attempting to reschedule each individual event, the improvement can be found in a fraction of the time that it would take to perform a full evaluation. This is essential in the case of this algorithm due to the sheer number of times that the operator is applied.

A deterministic Hill-Climbing algorithm is used at present. While it is considered generally better to incorporate as much non-determinism as possible in Hill-Climbing methods, the deterministic approach can be justified in this case as it only forms part of the solution finding mechanism. The algorithm can be summarised as follows:

- Take each event *e* in the order they appear in the timetable.
  - For each period *p* in the timetable, calculate the penalty that would arise from scheduling event *e* in period *p* if no hard constraints are broken.
  - Schedule *e* in the period causing least penalty.
  - Periodically try to schedule events from the list of unscheduled events.

2.3.4. The Evaluation Function

In order to ascertain the value of a solution the evaluation function is applied to it to calculate its *fitness*, and therefore its ability to survive in the total population. For the trials shown below the objective was to schedule all events in the given number of periods such that if a student has two exams on the same day then there should at least be a full period between them. The evaluation function can be expressed as:

$$\frac{10 numEvents}{2000 numUnscheduled + \sum_{i=0}^{i=numPeriods-1} NumConflicts(period_i, period_{i+1})}$$

where *numEvents* is the number of events in the timetable, *numUnscheduled* is the number of events not scheduled in a period yet, and *NumConflicts* is a function returning the number of conflicts, weighted by student numbers, between two periods on the same day.

2.3.5. The Selection Method

After application of the operators the population expands to a specified size. In order to reduce the population to its original size individuals are chosen from the pool to join the new population. This is achieved by using classic roulette wheel selection, which involves creating a roulette wheel that each individual has a section of that is proportional to his fitness relative to its competition. The wheel is then spun a number of times equal to the population size to select each individual member of the population.

3. Results

The algorithm has been tested on a range of real data in order to ascertain it's potential. Furthermore in order to ascertain the value of combining local search with the mutation operators in a population a multi-start random descent algorithm was run on the Nottingham data.

3.1. Nottingham University 1994/95 Data

The algorithm has been tested on the Nottingham University 1994/95 Semester one examinations data. This consists of 805 exams with 10,034 student conflicts between them, 7,896 students and 34,265 different enrolments. These exams must be scheduled into 32 periods or less such that the following constraints are met:

(1) No student is scheduled to take two exams at any one time.

(2) The maximum number of seats of 1550 is not exceeded.

(3) If a student is scheduled to take two exams in any one day there must be a complete period between the two exams.

The first two are, of course, the fundamental constraints that a timetable must satisfy in order to be at least feasible.

While this data is not as conflicting as it could potentially be, the scale of the problem presents a challenge to find a complete solution within a reasonable number of trials. The problem is also particularly compounded by the number and size of rooms available for each period. The algorithm was run with a population size of fifty on the same data varying the number of periods the timetable had to be scheduled in. The tests range from 23 periods (the minimum length given rooms constraints) to 32 periods (the time usually allocated by Nottingham University). Table 1 shows the results obtained. To give an idea of the CPU time involved the 32 periods case took roughly 3 hours to run to completion on a mid range Sun SPARC station, with all cases finding results within the time of an overnight run.

Periods	Evaluations	Num Unscheduled	Violations of *(3)*
32	1850	0	0
31	5450	0	0
30	12600	0	0
26	15500	0	49
23	9950	0	348

Table 1 - Results on Nottingham 1994/95 Data

To give some comparison the Nottingham data was also used to test how well the algorithm compared against a straight-forward random descent algorithm.

This consists of randomly generating a timetable then making random improvements until a number of tries have elapsed in which no improvements could be found. A new timetable is then generated and the process repeated, keeping the best timetable found throughout the process. Table 2 shows the results of this

when run on the Nottingham data for various numbers of periods for 8000 trials.

Periods	No. Unscheduled	Violations of *(3)*
32	0	14
31	0	33
30	0	66
26	0	214
23	3	752

Table 2 - Results of Applying Multi-Start Random Descent

The Nottingham data is freely available over the Internet from

`ftp://ftp.cs.nott.ac.uk/ttp/Data/`.

3.2. Other Data Sets

The following results are from data that is also freely available over the Internet from `ftp://ie.utoronto.ca/mwc/testprob`. The same constraints that were applied to the Nottingham data were applied to all of these problems, with the exception of varying the maximum number of seats per period to represent the real life maximum. Table 3 list the characteristics of each of the data sets while Table 4 lists the results when the algorithm was applied to each. The data used is represented by the following codes:

cars91	Carleton University 1991
carf92	Carleton University 1992
kfu	King Fahd University
tre	Trent University
utas	University of Toronto, Arts & Science

These data sets provide a reasonable range of benchmark problems for comparison with other methods.

Data	No. Exams	No. Students	No. Enrollments	Max Students per period
cars91	682	16926	56242	1550
carf92	543	18419	55189	2000
kfu	461	5439	25113	1955
tre	261	4362	14901	655
utas	638	21329	59144	2800

Table 3 - Characteristics of Data Used

Data	Periods	Evaluations	No. Unscheduled	Violations of *(3)*
cars91	40	6050	0	1232
carf92	33	10050	0	964
kfu	18	15750	0	1384
tre	30	7250	0	129
utas	35	8550	0	2226

Table 4 - Results on Data Sets

4. Conclusion

This algorithm offers an interesting approach to evolutionary timetabling. The addition of the Hill-Climbing operator after each mutation operator greatly increases the speed at which better solutions are found above the evolutionary operators alone. While the application of Hill-Climbing after each operation

does involve some computational expense, this is reduced by the use of delta-evaluation[10] and can be compensated for by the simplicity of the mutation operators, with any remaining net increase being justified by the reduction in total evaluations required.

Comparing the results on the Nottingham data with those obtained by applying multi-start random descent it appears that even though the random descent method was given significantly more CPU time it could not achieve equal results. Particularly notable is the fact that random descent could not find a perfect solution even when given the full 32 periods. This significant difference in results offers justification for the use of memetic algorithms and the operators within it.

Although the initial results are promising, the results have shown that the algorithm does not perform quite as well on the more highly constrained problems as some other methods[4] Therefore further research will be directed at refining the existing operators and introducing new ones. Experimentation with various methods of recombining different solutions found by the existing operators is of particular interest. This may include hybrid heuristic recombination operators such as those presented in [11] . Effort will also be directed at applying the algorithm to the lecture timetabling problem in order to investigate its potential on more general timetabling problems.

References

1. Burke E.K., Elliman D.G., and Weare R.F., ‘‘A University Timetabling System Based on Graph Colouring and Constraint Manipulation,’’ *Journal of Research on Computing in Education*, 1993.
2. E. K. Burke, D. G. Elliman, and R. F. Weare, ‘‘Examination Timetabling in British Universities - A Survey,’’ *To appear in the proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (Napier University, Edinburgh, UK)*, 30th Aug - 1st Sep 1995.
3. Davis L., ‘‘Handbook of Genetic Algorithms,’’ *Van Nostrand Reinhold*, 1991.
4. E. K. Burke, D. G. Elliman, and R. F. Weare, ‘‘A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems,’’ *6th International Conference on Genetic Algorithms (Pittsburgh, USA)*, 15-19 July 1995.
5. Dave Corne, Peter Ross, and Hsiao-Lan Fang, ‘‘Fast Practical Evolutionary Timetabling,’’ *Lecture Notes in Computer Science 865*, pp. 250-263, Springer-Verlag, 1994.
6. Paechter B., Cumming A., and Luchian H., ‘‘The Use of Local Search Suggestion Lists for Improving the Solution of Timetabling Problems with Evolutionary Algorithms,’’ *Lecture Notes in Computer Science 993 (Evolutionary Computing)*, pp. 86-93, Springer-Verlag, 1995.
7. Peter Ross and Dave Corne, ‘‘Applications of Genetic Algorithms,’’ *AISB Quarterly*, no. 89, pp. 23-30, 1994.
8. Pablo Moscato and Michael G. Norman, ‘‘A “Memetic” approach for the travelling salesman problem — implementation of a computational ecology for combinatorial optimisation on message–passing systems.,’’ *Proceedings of the International Conference on Parallel computing and Transputer Applications*, IOS Press (Amsterdam).
9. Nicholas J. Radcliffe and Patrick D. Surry, ‘‘Formal Memetic Algorithms,’’ *Lecture Notes in Computer Science 865 (Evolutionary Computing)*, pp. 1-16, Springer-Verlag, 1994.
10. Peter Ross and Dave Corne, ‘‘Improving evolutionary timetabling with delta evaluation and directed mutation,’’ *Parallel Problem Solving in Nature*, no. III, pp. Springer-Verlag, 1994.
11. E. K. Burke, D. G. Elliman, and R. F. Weare, ‘‘Specialised Recombinative Operators for Timetabling Problems,’’ *To appear in Lecture Notes in Computer Science (AISB Workshop on Evolutionary Computing)*, pp. Springer-Verlag, April 1995.