Fast Practical Evolutionary Timetabling

Dave Corne, Peter Ross
and Hsiao-Lan Fang

DAI Research Paper No. 708

To appear in Proceedings of the AISB Workshop on Evolutionary Computation, Springer Verlag, 1994.

Fast Practical Evolutionary Timetabling

Dave Corne, Peter Ross, Hsiao-Lan Fang

Department of Artificial Intelligence, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, U.K.

Abstract. We describe the General Examination/Lecture Timetabling Problem (GELTP), which covers a very broad range of real problems faced continually in educational institutions, and we describe how Evolutionary Algorithms (EAs) can be employed to effectively address arbitrary instances of the GELTP. Some benchmark GELTPs are described, including real and randomly generated problems. Results are presented for several of these benchmarks, and several research and implementation issues concerning EAs in timetabling are discussed.

1 Introduction

A number of researchers have applied evolutionary algorithms (EAs) to timetabling problems [2, 1, 7, 3, 8, 9]. Work so far has however tended to be isolated, applying a range of techniques to disconnected problems with little cross-comparison. The intent of this paper is to fully set out the nature of the problems addressed in EA timetabling research, and to present a series of results on some real and randomly generated problems which form part of a benchmark set we are collecting, using the (so far) most successful variant of the 'direct' approach. These benchmarks will hopefully spawn further, focussed work in the EA timetabling arena.

Section 2 describes the general form of the timetabling problem addressed by researchers using EAs and/or other techniques. Section 3 then notes how an EA may be set up to address an arbitrary instance of such a problem. In Sect. 4, we describe various test problems, both real and random, and gives tables of results on these problems. Section 5 then discusses a variety of aspects of the general approach which are worth mentioning, and some summary and conclusions appear in Sect. 6.

2 Evolutionary Timetabling

A large class of timetabling problems can be described as follows. There is a finite set of events $E = \{e_1, e_2, \ldots, e_v\}$ (for example, exams, seminars, project meetings), a finite set of potential start-times for these events $T = \{t_1, t_2, \ldots, t_s\}$, a finite set of places in which the events can occur $P = \{p_1, p_2, \ldots, p_n\}$, and a finite set of *agents* which have some distinguished role to play in particular events (eg: lecturers, tutors, invigilators, ...) $A = \{a_1, a_2, \ldots, a_m\}$. Each event e_i can be regarded as an ordered pair $e_i = (e_i^l, e_i^s)$, where e_i^l is the length of event e_i (eg:

in minutes), and e_i^s is its size (eg: if e_i is an examination, e_i^s might be the number of students attending that examination). Further, each place can be regarded as an ordered pair $p_i = (p_i^e, p_i^s)$, where p_i^e is the *event* capacity of the place (the number of different events that can occur concurrently in this place), and p_i^s is its size (eg: the total number of students it can hold). The event capacity matters: two exams can take place in the same room simultaneously, but two lectures cannot. There is also an $n \times n$ matrix D of travel-times between each pair of places.

An *assignment* is an ordered 4-tuple (a, b, c, d), where $a \in E$, $b \in T$, $c \in P$, and $d \in A$. An assignment has the straightforward general interpretation: "event a starts at time b in place c, and with agent d". If, for example, the problem was one of lecture timetabling then a more natural interpretation would be: "lecture a starts at time b in room c, and is taught by lecturer d".

Given E,T,P, A, and the matrix D, the GELTP involves producing a timetable which meets a large set of constraints C. A timetable is simply a collection of v assignments, one per event. How easy it is to produce a useful timetable in reasonable time depends crucially on the kind of constraints involved. In the rest of this section, we discuss what C may contain.

2.1 GELTP Constraints

Different instances of GELTPs are distinguished by the constraints and objectives involved, which typically make many (or even all) of the s^{vmn} possible timetables poor or unacceptable. Each constraint may be hard (must be satisfied) or soft (should be satisfied if possible). Many conventional timetabling algorithms address this distinction inadequately: if they cannot solve a given problem they relax one or more constraints and restart, thus trying to *solve a different problem*. The kinds of constraints that normally arise can be conveniently classified as follows.

Unary Constraints Unary constraints involve just one event. Examples include: "The science exam must take place on a Tuesday", or "The Plenary talk must be in the main function suite". They naturally fall into two classes:

Exclusions : An event must not take place in a given room, must not start at a given time, or cannot be assigned to a certain agent.

Specifications : An event must take place at a given time, in a given place, or must be assigned to a given agent.

Binary constraints A binary constraint involves restrictions on the assignments to a pair of events. These also fall conveniently into two classes:

Edge Constraints : These are the most common of all, arising because of the simple fact that people cannot be in two places (or doing two different things) at once. A general example is: "event x and event y must not overlap in time". The term 'edge' arises from a commonly employed abstraction of simple timetabling problems as graph colouring problems [12].

Juxtaposition Constraints : This is a wide class of constraints in which the ordering and/or time gap between two events is restricted in some way. Examples include: "event x must finish at least 30 minutes before event y starts", and "event x and event y must start at the same time".

Edge constraints are of course subsumed by juxtaposition constraints, but we single out the former because of their importance and ubiquity. Edge constraints appear in virtually all timetabling problems, and in some problems they may be the *only* constraints involved.

Capacity Constraints Capacity constraints specify that some function of the given set of events occurring simultaneously at a certain place must not exceed a given maximum. Eg: in lecturing timetabling we must usually specify that a room can hold just one lecture at a time. In exam timetabling this capacity may be higher for many rooms, but in both cases we need also to consider the total student capacity of a room. We may allow up to six examinations at once in a given hall, but only as long as that hall's maximum capacity of 200 candidates is not exceeded.

Event-Spread Constraints Timetablers are usually concerned with the way that events are spread out in time. In exam timetabling, for example, there may be an overall constraint of the form "A candidate should not be expected to sit more than four exams in two days". In lecture timetabling, we may require that multiple lectures on the same topic should be spread out as evenly as possible (using some problem-specific definition of what that means) during the week. Event-spread constraints can turn a timetabling problem from one that can be solved easily by more familiar graph-colouring methods into one which requires general optimisation procedures and for which we can at best hope for a near optimal solution.

Agent Constraints Agent constraints can involve restrictions on the total time assigned for an agent in the timetable, and restrictions and specifications on the events that each individual agent can be involved in. In addition to those already discussed (exclusions and specifications) we typically also need to deal with constraints involving agnet's preferences (for teaching certain courses, for example), and constraints involving teaching loads.

3 Applying EAs to the GELTP

In applying an EA to a problem, central considerations are the choice of a chromosome representation and the design of the fitness function. In this section we describe the approach we have found most successful so far.

3.1 Representation

For the GELTP, a chromosome is a vector of symbols of total length $3v$ (where v is the number of events), divided into contiguous chunks each containing three genes. The three alleles in the ith chunk, where $1 <= i <= v$, represent the time, place, and agent assignments of event i. Naturally, the set of possible alleles at time, place, and agent genes are respectively identified with the sets T, P, and A. The simple example chromosome "abcdef" represents a timetable in which event e_1 starts at time a in place b, involving agent c, and event e_2 starts at time d in place e, involving agent f.

Very many timetables thus represented will involve edge-constraint violations ('clashes'). Eg, the chromosome "abcabcabc......" is well-formed, even though it puts every event in the same place at the same time and involving the same agent. The job of the EA is to gradually remove such violations of constraints during the artificial evolutionary process.

3.2 The Evaluation function

It is important to be able to differentiate the relative quality of different timetables. An apparently satisfactory solution is widely used in the EA literature: it is simply to have fitness inversely proportional to the number of constraints violated in a timetable with each instance of a violated constraint weighted according to how important or not it is to satisfy it.

Let C_j be the set of constraints of type j (for example, event-spread constraints). The specific type classification used can be tailored to suit the problem. Each violated member of C_j attracts a specific penalty w_j. For each $c \in C_j$, let $v(c,t) = 1$ if c is violated in timetable t, and $v(c,t) = 0$ if c is satisfied. A simple fitness function for a GELTP is thus:

$$f(t) = 1/(1 + \sum_{\text{types}_j} w_j \sum_{c \in C_j} v(c,t)) \tag{1}$$

Assuming that all the penalties are positive, this function is 1 if and only if all the constraints hold, otherwise it is less than 1. The general idea is that an appropriate choice of values for the penalty terms should lead both to reasonable tradeoffs between different kinds of constraint violations, and (by virtue of defining the shape of the fitness landscape) effective guidance of the EA towards highly fit feasible timetables.

The approach works best if the constraint set C is fine grained. For example, if C contained only the 'single' constraint: "the timetable has no clashes", then the fitness landscape would contain a few spikes in an otherwise flat space, which is quite intractable to any form of search. C is hence best composed of low order constraints, each of which involves only one or two events. For example, the important constraint "No lectures which share common students should clash" appears in C as a large collection of separate constraints each involving a distinct pair of lectures which (are expected to) share common students or teachers. The set of such edge constraints are usually the largest single block of constraints in a timetabling problem. Commonly, applications will be faced with data in a form like: "student Jones sits exams Maths, Physics, Chemistry, ..."; this is then easily transformed to collections of binary constraints between events. In this case, each distinct pair of exams taken by the same student constitutes an edge constraint; typically, we may also automatically create an event-spread constraint between the same pair.

Given such a collection, a question arises as to how to treat them: they may either be treated as a uniform collection of distinct edge constraints of identical importance, or each edge may be weighted according to how many students share these events. In this way, and typically throughout this 'problem transformation' process, the constraint satisfaction problem (CSP) of finding a timetable which satisfies all the constraints can be transformed into any of several different constrained optimisation problems (COPs)[4]; each such COP will share at least one global optimum with the CSP (and preferably all of them) but will be otherwise different. In general, it seems important to make the landscape of this COP as meaningful as possible. For example, in an exam timetabling problem in which we treat each distinct edge constraint the same and each distinct event-spread constraint the same (but typically with event-spread constraints being less important edge constraints), we may find that the best we can do is find a collection of answers which violate a single event-spread constraint. These may be markedly different however; although each answer has just one pair of edge-constrained events timetabled too closely, some may involve only one student, while others may involve several. By weighting edge-constrained pairs of events (in this case according to the number of students sharing the given exams), the COP becomes more meaningful in that it is able to distinguish between such cases. This is particularly important in cases where the CSP has no solution, or where its solution is difficult to find; one or other of which is quite common norm in exam and lecture timetabling problems. In such cases, the COPs addressed by the EA (or some other method) may have different global optima, and so it becomes particularly important to use as 'meaningful' a COP as possible. In an example similar to that just discussed, for example, the less meaningful COP may have an optimum which violates just one event-spread constraint (which involves 50 students suffering consecutive exams, say), while a global optimum of the more meaningful version may involve just 2 students suffering consecutive exams, although having violations of two distinct event-spread constraints.

3.3 Speed

An important general consideration is that calculation of fitness be fast. Fortunately, this is usually true for most of the constraints we need to consider in the GELTP, which mainly comprise unary and binary constraints. More to the point, however, when using the direct representation it is particularly easy to set up 'delta evaluation', whereby to evaluate a timetable we need only consider the changes between it and an already-evaluated reasonably similar one such as one of its parents.

[9] discusses the use of delta evaluation further, noting that it is slightly more than just an obvious speedup measure. In particular, [9] notes that speed comparisons made in terms of 'number of evaluations', as commonly done, may often be overturned when delta evaluation is in use. That is, EA configuration X might regularly find results in fewer evaluations than EA configuration Y (and hence be faster when *full evaluation* is in use, but Y may be found to be faster than X when delta evaluation is employed. The reason for this is just that evaluations when using X in conjunction with delta evaluation generally take longer than with Y, because, for example, X employs a highly diversifying recombination operator which means that delta evaluation has to consider several changes each time (though will typically still be faster than full evaluation). It is important to note here that delta evaluation leads to the notion of *evaluation equivalents* EEs, which we employ later on. This simply records the time taken for a run in terms of the number of *full evaluations* that would have been done in the same time. We measure this, for example, by dividing the total number of constraint checks made during a delta evaluation run by the (constant) number that would be made during a full evaluation. This measure hence allows a machine independent measure of the time taken by an EA/timetabling run employing delta evaluation. An alternative is simply to record the total number of constraint checks made, but EE's provide a more accessible measure and give a more reasonable indication of total time taken.

3.4 Penalty Settings

Clearly, we can choose penalty terms for different constraints according to our particular idea of how we would trade off the advantages and disadvantages of different solutions to the problem in hand. Penalties must be set with care, however. If the ratio between two penalties (say, ordering constraints *vs* event-spread constraints) is too high, then search will quickly concentrate on a region of the space low in violations of the more penalised constraint, but perhaps missing a less dense region in which better tradeoffs could be found. If too low, then the capacity for the search to trade off between different objectives is lost. Evidently, optimal penalty settings depend on many things, primarily involving the density of regions of the fitness landscape in relation to each constraint, as well as the subjective relative disadvantages of different constraint violations in the problem at hand. Alternative possibilities include the approach in [10] (in the context of multiple objective facility layout problems), in which penalty settings are revised dynamically in accordance with the gradually discovered nature of the constrained fitness landscape. Also, a principled method for constructing scalar functions for multi-objective problems is discussed in the context of EA optimisation in [6]. In this method, called MAUA (Multi-Attribute Utility Analysis), extensive questioning of an expert decision maker (in our case, an experienced timetable constructor) on example cases (pairs of distinct timetables) would lead to the construction of a nonlinear function M of the vector of summed constraint violations, designed so as to best match the judgement of the expert in that the ordering on timetables imposed by M optimally matches that imposed by the expert. The effort involved in performing MAUA, however, is unlikely to be rewarded with a function M which is significantly closer to the expert decision-maker's judgement than an essentially *ad-hoc* but intuitive linear weighted penalty function and it is not clear that this is desirable. MAUA involves no attempt to structure M such that the fitness landscape is more helpful to the search process.

It suffices to say here that extensive experience so far suggests that for a wide range of problems we can settle for a simple linear penalty-weighted sum of violations with an intuitive choice of penalty settings.

3.5 Violation Directed Mutation Operators

In [9], a family of Violation Directed Mutation VDM operators for timetabling problems are examined, and it is found that a certain subclass of variations on VDM are particularly powerful for use on a range of realistic problems. Similar operators are studied in [4] for graph colouring and other constraint satisfaction problems. Here we adopt the use of one particular VDM variant, called `(rand, tn10)`, as standard. The action of this operator is roughly as follows (fuller description appears in [9]): an application of `(rand, tn10)` to a timetable amounts to randomly choosing an event (gene), and then selecting a new allele (timeslot) for it via tournament selection with a tournament size of 10; an allele's 'fitness' for this purpose is a measure of the degree to which it will reduce the degree of constraint violation involving the chosen event. This 'allele choice' operation involves some computational expense, but since the substantial part of it involves numbers of constraint checks, the time it takes can be meaningfully absorbed into our EE measure.

4 Some Benchmark Timetabling Problems

We first look at five real examination timetabling problems, and later consider 32 randomly generated highly over-constrained test problems. These are not fully general, in the sense of having a full repertoire of place and agent constraints as well as several kinds of timeslot constraint, but only consider edge, event-spread, exclusion, and, in the case of two of the real problems, timeslot capacity constraints (arising from a limit on the number of seats available in examination halls at any one time). Realistic fully general benchmarks will appear anon, but for now it seems reasonable to provide more simply defined problems (and hence more accessible for comparative performance research) which are nevertheless realistically difficult and/or common GELTP variants..

4.1 Some Real Examination Timetabling Problems

Three of these arise from MSc examinations at the EDAI[1], and two from Kingston University, London. Each of the EDAI problems, named in turn: `edai-ett-91`,

[1] University of Edinburgh Department of Artificial Intelligence

`edai-ett-92`, and `edai-ett-93`, involve a four slots per day timetable structure, and involve a number of edge constraints and exclusions. In each case, there is an event-spread constraint as follows: if any pair of edge-constrained events are timetabled to appear on the same day, then they must have at least one full slot between them. That is, they must occupy the first and third, first and fourth, or second and fourth slots. `edai-ett-91` has 314 edge constraints, and no exclusions, and must be timetabled over six days (hence 24 slots). `edai-ett-92` has 431 edges, no exclusions, and must occupy seven days, while `edai-ett-93` has 414 edges, 480 exclusions, and must occupy nine days.

The Kingston University problems respectively represent the first and second semester exams faced by Kingston University students in 1994. In the first semester problem, `ku-ett-94-1`, 97 exams have to be arranged over 5 days with with 3 slots per day. There are 399 edge constraints, and hence 399 individual event spread constraints. The event-spread constraint in this case is that if an edge constrained pair of exams both occur on the same day, they must occupy the first and third slot. `ku-ett-94-2` is the second semester problem; 128 exams must be arranged over an 8 day period, with 3 slots per day. The event spread objective this time is to avoid a student facing more than one exam per day. Hence, edge-constrained pairs of events should not occupy slots on the same day. There are 536 edge constraints, and hence 536 event spread constraints too. An additional constraint faced by both of the Kingston University problems is that a maximum of just 470 candidates can be seated in any timeslot. Hence, associated with each event is a weight representing the number of students taking the appropriate exam. These weights are then used by the fitness function (as well as by the VDM operator) to penalise (avoid) violations of this capacity constraint.

4.2 Default EA Configuration

The EA configuration used in all experiments was as follows. A reproduction cycle consisted of a breeding step (in which one new chromosome was produced) followed by an insertion step, in which this new chromosome replaced the currently least fit individual (but only if the new individual was fitter). With probability 0.2, a breeding step involved the selection of one parent and the simple gene-wise mutation of it with a probability of 0.02 of randomly reassigning the allele of each gene in turn. With probability 0.8, a breeding step involved the selection of one parent, and the application of the VDM operator `(rand, tn10`. Tournament selection was used with a tournament size of 6, and the population size was always 1,000.

Using the default configuration, we examine the reliability of this EA on five real timetabling problems. The default configuration was applied in 100 separate trials to each of the five problems detailed above. We record, in each case, the number of such trials which found an optimum (the 'number of perfect trials' column; on each of these problems, optima violating no constraints exist), the least, average, and most evaluations taken to reach an optimum for those trials which did, and the the least, average, and most evaluation equivalents taken to reach an optimum for those trials which did. These results appear in Table 1. Each trial on an `edai` problem was run for 25,000 evaluations, while trials on the `ku` problems ran for 40,000 evaluations each.

Table 1. Performance on five real lecture timetabling problems

	No. Perfect	Evaluations			Eval. Equiv's		
Problem	Trials	Lowest	Mean	Highest	Lowest	Mean	Highest
edai-ett-91	84	4595	7693	24933	4793	8084	26241
edai-ett-92	50	7701	14087	21703	5000	13771	21179
edai-ett-93	98	6611	9592	14818	4717	8501	13166
ku-ett-94-1	93	10273	14890	20243	4227	6181	8420
ku-ett-94-2	57	12594	19241	28808	3731	6109	9103

It may first be pointed out that these results show great general potential for EAs on timetabling problems. All trials are relatively fast; for example. Speed on the problems above ranged 250 to 400 evaluations per second[2] Even in the two cases where the EA found an optimum only 50% or so of the time, this means that a small number of trials would be more or less guaranteed to stumble on a perfect timetable soon enough. In the case of the `edai-ett-91` and `edai-ett-92`, the timetables actually used for these problems were independently produced by the relevant course administrators. As detailed in [3], these were very poor in relation to average EA-produced timetables on the same problems, and certainly far from the perfect timetables that the EA used here can typically find. From the `edai-ett-93` case on, the course administrators have been (thankfully) using an EA to do their timetabling work, and hence we do not have independently produced efforts for comparison. In the case of the `ku` problems, the course administrators at Kingston University tried to manually produce timetables for these problems but were having great difficulty, owing in particular to the recent modularisation of the underlying course; they also note that satisfying the capacity constraint in each case was particularly troublesome. Kingston University's timetablers notified us of their problems halfway through their troubled attempts, and sent us the relevant data, whereupon the EA managed to find perfect results regularly in each case.

The main intent in presenting these results is to provide benchmarks for future comparisons with other techniques. As detailed at the end of the paper, all problems we use are freely available. Comparative performance results on the above problems will focus on the *reliability* figure. That is, in looking for improvements on our timetabling EA, we are looking for a single method which will improve reliability in finding the optimum over all of the above set (and others). without any major speed sacrifice. Alternative valid targets include achieving

[2] On a Sun Sparcstation 2, using a not-necessarily optimised C program.

similar reliability faster, or perhaps less (but >50%, say) reliability but *much* faster.

4.3 Some Random Over-Constrained Timetabling Problems

A further kind of target, as is commonly the case with benchmark job shop scheduling problems, for example [11], is find increasingly lower bounds on the total penalty values for a suite of problems. Here we describe some timetabling problems involving edge and event-spread constraints which are constrained enough for this purpose, and present our best results so far using the EA described, and running to a limit of 200,000 evaluations in each of 10 trials for each problem. Each of these problems involves the same temporal structure and event-spread constraint as the `edai-ett` problems; that is, there are four timeslots per day, and for each edge constrained pair of events there is also an edge constraint which specifies that there should be at least one entire slot between them these events if they appear on the same day.

Each problem is named `bench-ett(X,Y,Z,W)`, where:

- X is simply an identifier of the set of random edge constraints involved.
- Y is the number of events to be timetabled.
- Z is the number of days allowed; hence there will be $4 \times Z$ slots altogether.
- W is the number of edge constraints. Each edge constraint is a pair of events, (e_1, e_2), constrained not to appear in the same timeslot. Associated with each constraint is the `edai-ett` style event-spread constraint, Further, associated with each edge constraint c in the set of edge constraints C is a weight c_w. This is an integer between 1 and 100 inclusive.

In each case, the EA applied a penalty P to a candidate timetable t as follows, in which $v(c,t)$ for $c \in C$ is 1 (0) if edge constraint c is violated (satisfied) in timetable t, while $e(c,t)$ for is 1 only if the event-spread constraint is violated but the corresponding edge constraint is *not* violated.

$$P(t) = \Sigma_{c \in C} 2v(c,t)c_w + e(c,t)c_w \qquad (2)$$

Figures in the second column of Table 2 are hence the minimal value for P we have so far found on these problems. The third and fourth columns respectively give the weighted penalty for edge constraints violated by the best timetable found, and the weighted penalty for event-spread constraints violated in the same timetable.

The problems in Table 2 are extremely highly constrained; almost certainly more so than any real timetabling problem. Nevertheless, comparative performance of techniques (EA-based or not) on this set (and similar sets) of problems will be useful because these are indeed timetabling-style problems, similar in the nature of the constraints involved than a very common class of real exam timetabling problems. Hence, better performance on these benchmarks will almost certainly reflect potential for better performance on real timetabling problems. Also, since these are very highly constrained COPs, it seems likely that

Table 2. Performance on Very Highly Constrained Random Timetabling Problems

Problem	Smallest Penalty	Edge Penalty	Event-Spread Penalty
bench-ett(1,50,5,1000)	1116	212	692
bench-ett(1,50,6,1000)	613	81	451
bench-ett(1,50,7,1000)	248	17	214
bench-ett(1,50,8,1000)	79	2	75
bench-ett(2,50,5,1000)	1011	174	663
bench-ett(2,50,6,1000)	477	57	363
bench-ett(2,50,7,1000)	151	15	121
bench-ett(2,50,8,1000)	84	7	70
bench-ett(3,50,5,1000)	1129	180	769
bench-ett(3,50,6,1000)	550	65	420
bench-ett(3,50,7,1000)	205	24	157
bench-ett(3,50,8,1000)	61	3	55
bench-ett(1,100,10,4000)	1359	159	1041
bench-ett(1,100,11,4000)	825	125	575
bench-ett(1,100,12,4000)	479	49	381
bench-ett(1,100,13,4000)	312	29	254
bench-ett(2,100,10,4000)	1492	265	962
bench-ett(2,100,11,4000)	1025	158	709
bench-ett(2,100,12,4000)	620	60	500
bench-ett(2,100,13,4000)	346	44	258
bench-ett(3,100,10,4000)	1493	331	831
bench-ett(3,100,11,4000)	874	170	534
bench-ett(3,100,12,4000)	400	46	308
bench-ett(3,100,13,4000)	232	35	162
bench-ett(1,500,1,5000)	122502	30619	61264
bench-ett(1,500,2,5000)	23576	3948	15680
bench-ett(1,500,3,5000)	4580	411	3758
bench-ett(1,500,4,5000)	453	48	357

there is great scope for *continual* improvement on the 'best so far' figures given above. That is, techniques better than the EA we have used so far may be found which achieve 100% reliability on the five real timetabling problems discussed earlier, and hence be indistinguishable on those problems. However, such techniques are likely to be separated in terms of their comparative performance on these highly constrained benchmarks, while still leaving room for further improvement. Lastly, being COPs with (almost certainly) no perfect solutions possible in each case, these problems reflect the difficulties involved in many real modular lecture timetabling problems; in such a problem, for example, a typical approach may be to attempt to allow *any* combination of courses as feasible in the timetable, to give students greater flexibility in their module choices. Since this goal is almost always infeasible, such a problem becomes a COP, with weightings on pairs of module choices reflecting the desirability of allowing these as viable combinations for a student.

5 Prospects for EAs in Timetabling

Indications so far point to the approach we have described as highly promising for general timetabling needs, at least of the kind usually found in educational institutions. However, several matters need pointing out with respect to its more general application. Firstly, as may be discerned, and as we have found, it is not always immediately apparent how best to describe the problem itself in terms of the kinds of constraints processed. Eg, to incorporate the overall event-spread constraint "no student should sit more than four exams in two successive days", one choice would be to have the objective function (partially) calculate each student's individual timetable, and directly penalise every instance of a student suffering the appropriate constraint violation. A potentially far less computationally expensive choice, however, might be simply to penalise all consecutive edge-constrained events which are less than, say, 2 hours apart in the overall timetable; though not addressing the constraint directly, this certainly applies the artificial evolutionary pressure in the right direction. These are just two of several possibilities for addressing this constraint. Generally, choices occur at many stages in the process of translating the constraints of a real GELTP into a penalty function, and much work is needed to discern the best way to do this. The difficulty may not be too great, however. At least *some* way of handling any given constraint will be naturally apparent, and, in our experience, it is unlikely that different choices will bring significantly different results, unless the problem instance is very large and hence solution speed is a major factor.

Another important aspect is comparative performance with other techniques. There is reason to suggest that the EA approach described will succeed more as a *general timetabling tool* than some other methods. In many cases, however, some other technique may be significantly preferable. Eg, a large, continually faced problem which changes little (in terms of the kinds of constraints involved) from instance to instance may be better off treated with some specifically designed algorithm. This may itself be based on an EA, involve an alternative EA-based approach, or be based on best-first search with a specifically designed heuristic, for example. In general, much work is required to discern what promises to be the best method for different kinds of GELTP.

Also, several aspects of the approach itself warrant much further study. Among these are the design of the objective function itself, as briefly discussed above and also in Sect. 3.4, the use of alternative operators, and various aspects of the underlying EA configuration.

Finally, timetablers (ie; human course organisers, for example) often have needs which are not directly met within the described approach. There may be some desire to generate several different possible timetables, for example, or there may be need for extensive preprocessing and altering of the constraints (which may initially contain several inconsistencies). Use of the EA described (via iterative applications, for example) may be useful as a tool in each case, but such considerations evidently require more useful refinements or extensions, if not other methods entirely. A planned extension to a future version of the EA described here, for example, is to have an ATMS-based constraint-checking front end. Also, experiments are under way which involve the generation of multiple distinct solutions in a single run, making use of EA variants specifically designed for multimodal optimisation.

6 Summary and Conclusions

We have described and noted various matters in the application of EAs to general educational-institution based timetabling problems. EAs seem to have great potential in this arena, and we illustrate this via presentation of various results. First, we show that a particular EA described here (but more fully in [9]), finds, quickly and reliably, perfect timetables for each of five real examination timetabling problems. The particular reliability results are offered as benchmarks against which to examine alternative techniques. Second, a collection of very highly constrained random timetabling-style problems are described, and our best results on these are given. Various research issues and considerations are then noted.

Finally, the test problems addressed may be freely obtained (and explained) from the authors, and/or via the FTP site ftp.dai.ed.ac.uk .

Acknowledgements

Thanks to Bob Fisher and Peter Sutcliffe for providing problem data. Thanks also to the UK Science and Engineering Research Council for support of Dave Corne via a grant with reference GR/J44513, and to the China Steel Corporation, Taiwan, R.O.C., for support of Hsiao-Lan Fang.

References

1. Abramson D., Abela, J. : A Parallel Genetic Algorithm for Solving the School Timetabling Problem. IJCAI workshop on Parallel Processing in AI, Sydney, August 1991
2. Colorni, A., Dorigo, M., Maniezzo, V.: Genetic Algorithms and Highly Constrained Problems: The Time-Table Case. Parallel Problem Solving from Nature I, Goos and Hartmanis (eds.) Springer-Verlag, 1990, pages 55–59
3. Corne, D., Fang H-L., Mellish, C.: 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.
4. Eiben, A.E., Raue, P.E., Ruttkay, Z Heuristic Genetic Algorithms for Constrained Problems. Working papers of the Dutch AI Conference, 1993, Twente, pages 341–353.

5. Corne, D., Ross, P., and Fang, H-L.: Fast Practical Evolutionary Timetabling. Proceedings of the AISB Workshop on Evolutionary Computation, Springer Verlag, 1994, to appear.
6. Horn, J., and Nafpliotis, N.: Multiobjective Optimisation Using The Niched Pareto Genetic Algorithm. Illinois Genetic Algorithms Laboratory (IlliGAL) Technical Report No. 93005, July 1993.
7. Ling, S-E.: Intergating Genetic Algorithms with a Prolog Assignment Problem as a Hybrid Solution for a Polytechnic Timetable Problem. Parallel Problem Solving from Nature 2, Elsevier Science Publisher B.V., Manner and Manderick (eds.), 1992, pages 321–329.
8. Paechter, B. Optimising a Presentation Timetable Using Evolutionary Algorithms. Proceedings of the AISB Workshop on Evolutionary Computation, Springer Verlag, 1994, to appear.
9. Ross, P., Corne, D., and Fang, H-L.: Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation.: Proceedings of PPSN III, Jerusalem, October 1994, Springer Verlag, to appear.
10. Smith, A.E., and Tate, D. M.: Genetic Optimisation Using a Penalty Function. Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo: Morgan Kauffman, S. Forrest (ed), 1993, pages 499–503.
11. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of operations research, Volume 64, 1993, pages 278–285.
12. Wilson, R. J.: Introduction to Graph Theory. Longman, London, 1979.

This article was processed using the LaTeX macro package with LLNCS style