Simultaneous Satisfaction of Hard and Soft Timetable Constraints for a University Department Using Evolutionary Timetabling

Damien Woods and Adrian Trenaman

Department of Computer Science,
National University of Ireland, Maynooth,
Co. Kildare,
Ireland.
Phone: +353-1-7083354 Email: trenaman@cs.may.ie

Abstract. An evolutionary approach is applied to the problem of designing timetables in a university setting. An indirect encoding is employed, where the structures under evolution are lists of instructions for creating timetables. Timetables are evaluated with a fitness measure that captures both hard constraints (for example, no resource clashes) and soft constraints (where timetables that cause less "pain" to students and lecturers are favoured over those with more "pain"). The technique is applied to synthetic data and sample data from the Department of Computer Science at NUI Maynooth, and suitable timetables are produced with all hard constraints satisfied and soft-constraints minimised. A wide range of experiments indicates that relatively high mutation rates are beneficial with this approach.

1 Introduction

This paper applies an evolutionary approach to a real-world instance of the timetabling problem, using data from the Department of Computer Science at NUI Maynooth. Any automated timetabler must satisfy hard-constraints, that is, those constraints that must be satisfied for a timetable to be considered valid. For example, a lecturer cannot give two lectures at the same time, similarly, two lectures cannot take place at the same time in the same theatre. As an NP-complete problem isomorphic to the graph-colouring and packing problems, heuristic techniques exist to locate solutions that satisfy hard constraints [3, 1]. In this paper we consider also the satisfaction of *soft* constraints: these are constraints that are not essential for a valid timetable, but nonetheless capture what makes one timetable *better* than another. Soft constraints embody what is desirable in a timetable, for both lecturers and students, favouring lectures at certain times and minimising the number of lectures and events "in a row".

In this context, candidate timetables are measured in terms of their overall goodness as a combination of hard and soft-constraint satisfaction, rather than a simple binary valid / invalid measure. This "goodness" is interpreted as fitness, and an evolutionary algorithm is used to guide a population of randomly generated timetables towards a solution that maximises this fitness measure (for a good overview of the evolutionary approach, see [5, 4]). Previous evolutionary approaches to timetabling have employed either direct or indirect representations. In the former, the structures subject to the evolutionary process are timetables, while in the latter the structures are sequences of instructions used to build timetables through a developmental process. Although fast [2], the direct approach explores a space dense with invalid timetables. The indirect approach [6, 7], on the other hand, explores spaces with much fewer invalid timetables, but is slower as every timetable must be constructed before evaluation.

In this paper we employ an indirect evolutionary approach to the timetabling problem: the approach is described in Section 2. Experiments are carried out in Section 3 to examine the performance of the system, with results in Section 4. Conclusions are presented in Section 5: the approach successfully produces timetables with all hard-constraints satisfied and soft-constraints minimised. An exploration of the parameter settings reveals that high levels of mutation are appropriate with this system. Some ideas for future work are presented in Section 6.

2 Approach

The approach adopted here is an indirect encoding similar that of Paechter *et al.* [6]. In the next sections we describe the underlying representation, evolutionary operators and how the timetables are constructed and evaluated. Evaluation calculates fitness based on the timetable's satisfaction of hard and soft constraints, where soft-constraints are defined in relation to a novel *pain-table.*

2.1 Representation

The representation is oriented towards the events that must be placed in the timetable. In a university setting, an *event* is a lecture, tutorial or laboratory session to which time (timeslots), resources (rooms) and agents (lecturers or demonstrators) must be allocated. Every individual (chromosome) in the population consists of a fixed length list of genes, one for each event that must be placed. Each gene contains seven attributes, listed in Table 1.

Attribute	Description
agent_allocate	The agent that will be allocated to the event.
resouce_allocate	The resource that will be allocated.
timeslot_allocate	The timeslot allocated to the event.
timeslot_direction	What direction to search through the list of timeslots for a free slot, if allocation to timeslot_allocate results in a direct hard-constraint violation. This value is either *forward* or *backward.*
next_gene	A pointer to the next gene in the list to be allocated
gene_direction	The direction to search through the chromosome to find an unplaced gene if the current gene is already placed or unplaceable. This value is either *forward* or *backward.*
placement_status	The status of this event: placed, unplaceable or not-yet-placed.

Table 1. Each gene holds the instruction for the placement of a single event, and contains the above attributes.

Each chromosome is interpreted as a list of instruction on how to build a timetable. The process begins by randomly picking a gene in the chromosome and attempting to place the associated event in the timetable at the timeslot indicated by attribute *timeslot_allocate.* If placement of the event does not incur a hard constraint violation of agents or resources (a clash) then the timeslot is allocated the event, resource and agent represented by the gene. If this placement would result in a clash then the timeslot list is searched for the next free space. The attribute *timeslot_direction* is used to determine which direction this search should be made, either forward or backward. If the search fails, that is, there is no timeslot to which the event can be placed without a clash, then the event is marked as unplaceable.

After the attempted placement of an event, the value of attribute *next_gene* determines the next gene (event) that will be processed. If that gene has already been placed or is unplaceable (this can be determined by examining the *placement_status* attribute) then the next gene to be placed is determined using the *gene_direction* attribute, indicating which direction to search along the list for an unplaced gene to process. Using *gene_direction* in the event of placed or unplaceable genes ensures that all genes in the list will eventually be processed, and eliminates the problem of circular routes through the gene list.

To speed up subsequent timetable builds, whenever *gene_direction* is used to determine the next gene to process, the gene eventually chosen is written back into *next_gene* attribute of the last gene processed. So, the construction of a timetable (phenotype) from the chromosome (genotype) allows change to occur in the genotype, effecting Lamarckian learning.

2.2 Operators

Individuals in the population are randomly created by placing random values in the attributes of the genes. Operators used are reproduction, crossover and two kinds of mutation. Reproduction is the standard copying of an individual into the next generation. One-point crossover is performed by taking two individuals and picking a random position on the list. A new individual is then made up of the head of one list and the tail of the other. Mutation is performed on an individual by picking a single gene, and then randomly changing either one or all of the gene's attributes.

2.3 Evaluation

The fitness of an individual in the population is determined by using the encoded instructions to construct a timetable, using the algorithm outlined above. The timetable is then evaluated by examining the number of constraint violations and weighting the result. Let C be the set of constraints that must be adhered too. Then for each $c_j \in C$ there exists an associated weighting w_j. Also for each $c_j \in C$ let $v(c_j, t)$ be the number of times constraint c_j is violated in timetable t. Then the fitness of timetable t, $f(t)$, is defined as:

$$f(t) = \sum_j w_j v(c_j, t)$$

All weightings are positive, so fitness will be 0 when all constraints are satisfied (if that is possible) and greater than 0 otherwise. Appropriate choices for weights will lead to reasonable trade-offs between the different constraints that hopefully will guide the evolutionary algorithm towards good timetables.

We distinguish between two kinds of constraint: *hard* and *soft* constraints.

Hard Constraints Hard constraints are those constraints that *must* be satisfied by a timetable. Due to the approach used here to build the timetables, some typical hard-constraints cannot occur. That is, an event can never appear more than once in the table, and an agent or resource can never be scheduled more than once in a given timeslot. There are four hard-constraints that can occur in the scheme used here:

- **Unplaced events.** After construction of the timetable, there may be events left unplaced.

– **Student clashes.** Student clashes occur when a student is scheduled for more than one event at the same time.
– **Unqualified lecturers.** These occur when a lecturer is assigned an event that he or she is not qualifed to teach.
– **Inadequate facilities.** The room assigned to a particular event is too small or does not have wheelchair access (if required).

Soft Constraints Soft-constraints are those constraints that are not necessary for a valid timetable, but are required for a *good* timetable. The satisfaction of soft-constraints is what makes one valid timetable better than another. Many factors can be considered in the formulation of soft constraints, however if too many soft-constraints are employed then it may become impossible to satisfy all of them.

The soft-constraints treated in this work relate to the merit of a timetable from a student's or agent's perspective, with one exception. A placement is penalised if an event requiring a laptop projector is placed in a room with no projector. This was seen to be a soft rather than hard constraint as the lecturer could bring a portable projector if required.

	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00
Mon	0.9	0.8	0.7	0.3	0.5	0.3	0.2	0.4	0.6
Tues	0.7	0.6	0.5	0.0	0.3	0.0	0.1	0.2	0.5
Wed	0.7	0.5	0.2	0.0	0.3	0.0	0.0	0.2	0.5
Thur	0.7	0.5	0.0	0.0	0.3	0.0	0.0	0.3	0.5
Fri	0.8	0.7	0.5	0.3	0.4	0.5	0.7	0.8	0.9

Table 2. Pain Table: all values are between 0 and 1, where a higher value means greater pain.

The other soft-constraints focus on designing timetables that the students and lecturers of the university will find amenable, and are summed up in by the word "pain". In a university setting, a large body of people have their movements, work and play influenced by the timetable. Active participants tend to have (often very similar) opinions on what constitutes a good and a bad timetable. A survey of students and lecturers was carried out, where participants weighted timeslots in the week according to how painful they were, as a number from 0 to 50. This information was collated and the resulting pain-table is shown in Table 2. The pain of an event is then calculated using the associated pain-value, with consideration also towards the number of multiple events (a number of events in a row) taking place. Pain is calculated using the following equation:

$$p(t) = [\frac{2}{n^3 . p_{max}}].m^2.p_{accum}$$

where n is the number of timeslots, p_{max} is the maximum possible pain, m is the number of multiples and p_{accum} is the accumulative multiple pain values. The function $p(t)$ was designed to increase quadratically as m, the number of multiples, increases, and increases linearly with p_{accum}.

3 Experiments

Initial experiments were run using randomly generated synthetic data with 25 events, 10 agents and five rooms. Only three of the events were multiple, each being 2 timeslots long. This data set was small, random and loosely constrained, providing a "proof-of-concept". The system easily evolved timetables that satisfied all hard constraints within 100 generations.

The second data set was taken from real-world data from the Department of Computer Science at NUI Maynooth. There were 68 events in total, 43 of these being lectures of one timeslot in length, the others being lab sessions of 2 or 3 timeslots in length. There was a total of 7 distinct student groups. Seven of the lectures required attendance from two groups, the rest requiring attendance from just one. The lecture sizes varied from 28 to 312 students.

The laboratory sessions were classed as either "UNIX" or "Windows", depending on the type of machine they required. Lab event sizes varied from 18 to 81 students. This high number of lab sessions makes 37% of the events 2 or 3 timeslots long, making the problem hard for two reasons. First, students get "tied-up" in long events, leaving a higher chance of clashes when placing other events. Second, long events have fewer placement possibilities: for example, a 3-hour lab cannot be placed starting in the last two timeslots of the day.

The amount and type of agents available made this problem very highly constrained. In general only one lecturer can give a particular event. More freedom is allowed in the allocation of demonstrators for labs from a demonstrator "pool". A total of 16 agents could take the 43 lecture events, while a maximum of 16 other agents were available for lab sessions. By allowing the evolutionary process to perform the lecturing assignment (even though this may be fixed *a priori*) it can suggest alternative allocations in the event of clashes.

With only 43 lecturing events it is possible to place all events in just one lecture theatre: accordingly one large theatre was provided. Three labs, two "Windows and UNIX" and one UNIX lab were available. Their respective capacities were 81, 53 and 30 students.

Experiments were performed with two main goals: first to show that the system could discover timetables with no hard-constraint and minimal soft-constraint violations; and second, to examine how different rates of mutation, crossover and reproduction affect the evolutionary process. Small populations of size 100 were used, and the selection method was *tournament* selection over 7 individuals. Initially, 16 different rate-settings were identified, for example, "1% mutation, 90% crossover, 9% reproduction". Reproduction was generally left at around 10%, the other 90% spread between mutation and crossover. 5 runs were performed for each setting for 1250 generations. The best run of each category each was then rerun for 2000 generations. This gave an indication of the best settings for the operator rates: a number of different rate-settings in this region were identified and more runs carried out.

4 Results

Some average fitness curves from the final runs (those where the optimum operator rates had been identified, with mutation ranging from 34% to 90%) are shown in Figure 1. Convergence on good solutions occurred over a wide variety of the parameters within 1200 generations and in the best cases within approximately 500 generations. In general, high mutation gave the best results: with lower mutation and higher crossover premature convergence on poor solutions tended to occur. The best timetable was produced using 48% mutation, 42% crossover and 10% reproduction. However, when mutation was higher than 79% timetables were produced with some hard-constraint violations.

The evolutionary process successfully minimised the pain function, with resulting timetables filling low-pain timeslots and having few multiple events. Optimisation of pain began very early, typically in the first 10 generations. Examining one successful run, the last hard-constraints to be satisfied were generally the placement

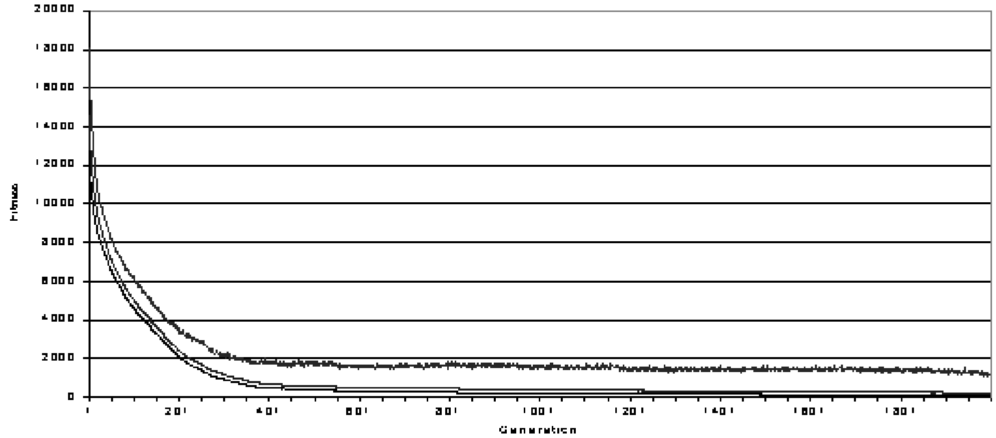

Fig. 1. Average fitness curves. The lower line is average *best* fitness, the middle is average mean-fitness, while the upper line is average worst-fitness.

of previously unplaced multiple events and assignment of agents. Event resource dependencies (e.g. incorrect room type, incorrect room size) were resolved quickly, with student group clashes resolved first.

Generally, the evolutionary process seems to place the atomic events first, along with whatever multiple events it can. It then proceeds to optimise positions for the atomic events (that is, reduce pain) while some multiple events remain unplaced. Then the timetabler juggles between disrupting the existing low-pain allocation and placing the unplaced multiple events.

5 Conclusions

In this paper we have described the design and performance of an evolutionary timetabling system allowing the simultaneous satisfaction of hard and soft timetable constraints, where soft-constraints are captured using a novel timetable "pain" metric. The system has been shown to produce clash-free low-pain timetables on real-world data, and works best when equal doses of mutation and crossover (at approximately 45% each) are used.

6 Future Work

The success of the system on departmental data prompts the question of whether the approach will scale, for example, to tackle the timetabling problem at a university level, where resources must be shared within different departments.

In a larger system complexity might be reduced by removing the agent allocation problem, as in most institutions this remains fixed for lecture events and can be hand allocated for laboratory sessions.

An investigation into mechanisms that promote the placing of multiple events in the earlier stages of evolution may speed up the satisfaction of hard-constraints.

References

1. E. K. Burke, D. G. Elliman, and R. Weare. A university timetabling system based on graph colouring and constraint manipulation.
2. D. Corne, P. Ross, and F. Hsiao-Lan. Fast practical evolutionary timetabling. In *Selected Papers: AISB Workshop on Evolutionary Computing, Lecture Notes in Computer Science No. 865*, 1994.
3. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 1985.
4. David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, New Jersey, USA, 1995.
5. David E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison Wesley, 1992.
6. B. Paechter, H. Luchian, and A. Cumming. Two solutions to the general timetabling problem using evolutionary methods. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 300–305, 1994.
7. D. Ross and D. Corne. Comparing genetic algorithms, simulated annealing and stochastic hillclimbing on timetabling problems. In *Proceedings of the AISB Workshop on Evolutionary Computing*, 1995.