EVOLUTIONARY ALGORITHMS TO SOLVE THE SCHOOL TIMETABLING PROBLEM

Fernández N., Alfonso H.
Proyecto UNLPAM-09/F015[1]
Departamento de Informática - Facultad de Ingeniería
Universidad Nacional de La Pampa
Calle 110 esq. 9
(6360) General Pico – La Pampa – Rep. Argentina
e-mail: { fernaty, alfonsoh }@ing.unlpam.edu.ar
Phone: +54 2302 422780/422372, Ext. 6302

Gallard R.
Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC)[2]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
(5700) - San Luis -Argentina
e-mail: rgallard@unsl.edu.ar
Phone: +54 2652 420823
Fax : +54 2652 430224

ABSTRACT

Evolutionary Computation (EC) have been applied to a great number of optimisation problems with some success, specially in fields such as production operations in manufacturing industry, parallel and distributed systems, logistics and traffic.
The Timetabling problem is a kind of three-dimensional cutting problems, but it can be considered as a kind of scheduling problems too. The problem of creating a valid timetable involves scheduling classes, teachers and rooms into a fixed number of periods, in such a way that no teacher, class or room is used more than once per period.
The EC mimics the process of natural selection, with the effect of creating a number of potentially optimal solutions to some complex search problem. This work shows an evolutionary approach to solve the school-timetabling problem. To ensure the birth of valid offspring this approach uses a population of decoders. Implementation details for a set of timetabling instances of distinct complexity, using evolutionary approaches to the problem, are shown.

[1] The Research Group is supported by the Universidad Nacional de La Pampa.
[2] The LIDIC is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

1. INTRODUCTION

The problem of creating a valid timetable involves scheduling classes, teachers and rooms into a fixed number of periods, in such a way that no teacher, class or room is used more than once per period. For example, if a class must meet twice a week, then it must be placed in two different periods to avoid a clash. A class consists of a number of students. We assume that the allocation of students into classes is fixed, and that classes are disjoint, that is, they have no students in common. For them, a correct timetable is one in which a class can be scheduled concurrently with any other class. In each period, a class is a subject. It is possible for a subject to appear more than once in a period. A particular combination of a teacher, a subject, a room and a class is called a tuple. A tuple may be required more than once per week. Thus, the timetabling problem can be phrased as scheduling a number of tuples such that a teacher, class or room does not appear more than once per period. There are several works in the literature that attempt to solve this problem using different heuristics [1,2, 3,6, 7].
The task of combining a class, teacher and room into a tuple is handled as a separate operation. Tuples are formed with knowledge of the requirements of the various classes and teachers, as well as information on room availability. It is convenient to partition the problem in this way as it reduces the complexity of the scheduling task. In many cases it is possible to form the tuples without the need to use an optimisation scheme because the relationship between classes, teachers and rooms is often fixed.
It is possible to define an objective or cost function for evaluating a given timetable [8]. This function is an arbitrary measure of the quality of the solution. A convenient cost function calculates the number of clashes in any given timetable [2]. An acceptable timetable has a cost of 0. The optimisation problem becomes one of minimising the value of this cost function. The cost of any period can be expressed as the sum of three components corresponding to a class cost, a teacher cost and a room cost. The sum of the components is not strictly necessary but it can reflect the quality of the solution.
However, by using a sum, it is easy to weight the various costs so that one may be made more important than the others. In this way the optimisation process can be guided towards a solution in which some type of clashes are more important than others.
The class cost is the number of times each of the classes in the period appears in that period, less one if it is greater than zero. Thus if a class appears no times or once in a period, then the cost of it is zero. If it appears many times, the class cost for that class is the number of times less one. The class cost for a period is the sum of all class costs. The same computation applies for teachers and rooms. The cost of the total timetable is the sum of the period costs. Therefore, any optimisation technique should aim to find a configuration with the lowest possible cost.

2. APPLYING EVOLUTIONARY ALGORITHMS TO TIMETABLING

A timetable can be represented by a fixed set of tuples, each of which contains a class number, a teacher number and a room number. The scheduling algorithm must assign a period number to each of these tuples such that a given class, teacher or room does not appear more than once in a period.
In order to use an evolutionary algorithm to solve this problem, it is necessary to devise a representation for a timetable which maps onto chromosomes. Each chromosome is composed of genes, each of which represents some property of the individual timetable.
The specification of an appropriate schedule representation is a decisive factor to find a good performance of an evolutionary algorithm [4,5]. The principal difficulty present in scheduling problems is to determine which chromosome representation is to be used. We choose an indirect chromosome representation for the scheduler, where the transformation from a chromosome representation to a legal schedule had been done for a special decoder. With this kind of representations, the algorithm works on a population of encoded solutions [9]. Because the representation do not directly provides a schedule, a scheduler builder is necessary to transform a chromosome into a schedule, validate and evaluate it. The scheduler builder guarantees the feasibility of a solution and its work depends on the amount of information included in the representation.

Decoders

Here a chromosome is an *n*-vector where the i^{th} component is an integer in the range 1..(*n*-*i*+1). The chromosome is interpreted as a strategy to extract items from an ordered list *L* and build a permutation. For example if *L* = (1, 2, 3, 4, 5, 6) then table 1, indicates the correspondence among chromosomes and permutations.

Chromosome	Permutation
1 1 1 1 1 1	1 2 3 4 5 6
4 3 4 1 1 1	4 3 6 1 2 5
4 2 3 2 1 1	4 2 5 3 1 6

Table 1. Chromosome-permutation correspondence

A decoder is a mapping from the representation space into a feasible part of the solution space, which includes mappings between representations that evolve and representation that constitute the input for the evaluation function. This simplifies implementation and produces feasible offspring under different conventional crossover methods, avoiding the use of penalties or repair actions.
Although the offspring was feasible, the schedule builder could not find a place in the timetable for some tuples then a repair action or penalty is necessary for this individual.

3. IMPLEMENTATION DETAILS

The experiments were developed for a set of selected instances of the Timetabling Problem, under permutation representation. For each experiment series of fifty runs were performed. Elitism to retain the best valued individual was implemented. The population size was fixed at 50 individuals. For crossover, we used the traditional *one-point* crossover operator. And for mutation, a *big-creep* operator was used, which consists in replacing the gene value by another in the permitted range. The algorithms evolved for a minimum of 500 generations, after that a control of the mean fitness population progress began: if this value remained within a determined range for 20 consecutive then the algorithm stops. Probabilities for crossover and mutation were fixed at 0.65 and 0.01, respectively. These values were determined as the best combination of probabilities after many initial trials. At the moment, we are testing the described algorithm on three instances [3]: *hdtt4*, *hdtt5* and *hdtt6*.

4. REFERENCES

[1] Abramson D. "Constructing School Timetables Using Simulated Annealing: Parallel and Sequential Solutions", Management Science, pp 98-113. Jan 1991.

[2] Abramson D., Abela J.: *A Parallel Genetic Algorithm for Solving the School Timetabling Problem*, 15 Australian Computer Science Conference, pp 1-11. Feb. 1992.

[3] Abramson D.A. and Dang H. ``School Timetables: A Case Study in Simulated Annealing", in Lecture Notes in Economics and Mathematical Systems: Applied Simulated Annealing, Rene V.V. Vidal (Ed.), 103-124, 1993.

[4] Bäck T: *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996.

[5] Bruns R.: *Scheduling*, Handbook of Evolutionary Computation, 1997, chapter F1.5, pp F1.5:1-F1.5:9. 1997.

[6] Burke E.K., Elliman D.G., Weare R.: *A Hibrid Genetic Algorithm for Highly Constrained Timetabling Problems*, Proceedings of the Sixth International Conference on Genetic Algorithms, 1995.

[7] Caldeira J.P., Rosa A.C.: *School Timetabling using Genetic Search*. Laseeb - ISR – IST. 1997.

[8] Goldberg D. E.: *Genetic Algorithms: In Search, Optimization and Machine Learning*. 1989, Addison-Wesly Publishing Co.

[9] Salto C., Alfonso H., Gallard R.: *Performance Evaluation of Evolutionary Algorithms under Different Representations when Solving the JSS problem*, Proc. of 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2001), pp. 424-428, 2001.