A GA evolving instructions for a timetable builder

Christian Blum[1], Sebastião Correia[2], Marco Dorigo[1], Ben Paechter[3], Olivia Rossi-Doria[3], and Marko Snoek[4]

[1] IRIDIA, Université Libre de Bruxelles
Brussels, Belgium
{cblum,mdorigo}@ulb.ac.be
[2] Chronopost International
Paris, France
s-correia@chronopost.fr
[3] School of Computing, Napier University
Edinburgh, Scotland
{o.rossi-doria,b.paechter}@napier.ac.uk
[4] Department of Computer Science, University of Twente
Twente, The Netherlands
snoek@cs.utwente.nl

Abstract. In this work we present a Genetic Algorithm for tackling timetabling problems. Our approach uses an indirect solution representation, which denotes a number of instructions for a timetable builder on how to sequentially build a solution. These instructions are composed by a set of predefined heuristics. The ongoing work presented in this abstract was started by the authors at the EvoNet summer school 2001.

1 Introduction

Timetabling problems are variants of the general resource allocation problem where resources have to be allocated to certain tasks. A university timetabling problem – as considered in this work – consists in assigning each class from a set of classes to a suitable room and a timeslot. Numerous metaheuristic algorithms have been proposed to tackle this highly constrained problem. Among them are Simulated Annealing (SA) approaches (e.g., [1]), Tabu Search (TS) methods (e.g., [7]), and quite a few approaches from the area of Evolutionary Computation (EC) (e.g., [3, 2]).
The Genetic Algorithm (GA) presented in this work searches the space of heuristic rules to be applied during a step–by–step construction of a timetable. The idea of evolving instructions – composed by a set of heuristic rules – for a construction mechanism to build solutions is inspired by the research on evolutionary algorithms for scheduling problems (e.g., [4, 5]). For scheduling problems, it resulted in very successful algorithms, successful in terms of solution quality, but also in robustness and flexibility.

2 Problem definition

The timetabling problem considered here is a reduction of a typical university timetabling problem. A problem instance generator produces problem instances with different characteristics for different values of given parameters. The problem consists of a set of events E to be scheduled in 45 timeslots (5 days of 9 hours each), a set of rooms R in which events can take place, a set of students S who attend the events, and a set of room features F required by events. Each student attends a number of events and each room has a size. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following hard constraints are satisfied:

- no student attends more than one event at the same time;
- the room is big enough for all the attending students and satisfies all the features required by the event;
- only one event is in each room at any timeslot.

In addition, a candidate timetable is penalized equally for each occurrence of the following soft constraint violations:

- a student has a class in the last slot of the day;
- a student has more than two classes in a row;
- a student has a single class on a day.

3 Our approach

We chose to use the usual framework of a genetic algorithm for our approach. This means, our algorithm works on a population of individuals, applying crossover (e.g., one–point or uniform crossover) and a simple mutation operator to them. As a selection scheme we chose tournament selection. In the following we outline the most important features of this algorithm, namely the specification of individuals, the timetable builder which takes an individual as input and produces a solution from it, and the specification of the fitness function.

3.1 Individuals

In order to reduce the complexity of the timetabling process, we chose a sequential approach. In this approach events are assigned consecutively to a room and then to a timeslot. This requires $|E|$ constructions steps, each one consisting of (i) choosing an unprocessed event $e \in E$, (ii) assigning a room $r \in R$ to e, (iii) assigning the pair $< e, r >$ to a timeslot. Individuals consist of a three row matrix. For each one of the three steps (i)–(iii) there are a number of priority rules available. The number of columns in the matrix is equal to the number of events $|E|$. Position j in the first row specifies the priority rule to choose for performing step (i) in the jth construction step mentioned above (the choice of an unprocessed event). Consequently, position j in the second row specifies the priority rule for performing step (ii) in the jth construction step, and position j in the third row specifies the priority rule for performing step (iii) in the jth construction step.

3.2 Heuristic priority rules

A number of heuristic priority rules for steps (i)–(iii) have been devised. For instance, for choosing a class to be processed next, priority can be given to classes with a high number of students attending. The reason for this is, that one would expect these classes to be problematic in the sense that they are likely to produce clashes when there are parallel classes (classes in the same timeslot). The following *class choice priority rules* have been devised. Among the unprocessed classes, choose the class

1) that has the highest number of students attending,
2) that requires a room which is most required according to the set of unprocessed classes.

For the *assignment of a room* to an already chosen event e, the following rules were devised. Choose among the rooms of correct room–type the

1) smallest possible room,
2) the room with the lowest utilization rate in the partial timetable (where the partial timetable is the current timetable that is under construction).

Using any of these rules will assure that every event will take place in a suitable room. So, these room assignment rules already take care of one of the two possible types of hard constraints. For *assigning an event–room pair to a timeslot*, we implemented the following priority rules. Choose among the timeslots which would cause the lowest number of clashes the one

1) which has the most parallel classes,
2) which has the most students attending a class in parallel.

In applying any of the rules above we use the following policy. If there are ties, we always take the event, room, or timeslot with the lowest label. This is done to make the process deterministic.

3.3 The timetable builder

Given an individual, the timetable builder can incrementally construct a timetable in the following, deterministic way. For construction steps $j = 1, ..., |E|$ do:

- Choose an unprocessed class $e \in E$ using the rule specified in the j-th position of the first row of the chromosome matrix.
- Assign class e to a room r using the rule specified in the j-th position of the second row.
- Assign the pair $< e, r >$ to a timeslot using the rule specified in the j-th position of the third row.

3.4 The fitness function

The evaluation of a timetable consists of two stages. First of all, the feasibility of a timetable is checked. A timetable is feasible if and only if there are no violated hard constraints. Secondly, after feasibility has been checked successfully, a preference for the timetable is calculated by counting the number of soft constraint violations. This number is called the score of the timetable. For use in an evolutionary algorithm, this score is inadequate, because we also have to allow unfeasible timetables. Therefore we chose the common approach of evaluating the fitness by summing up the weighted number of hard and soft constraints. These weights have to be carefully chosen.

4 Conclusions and outlook to the future

In this paper we have presented a Genetic Algorithm that evolves a policy for a timetable builder to construct good timetables. The availability of powerful heuristic priority rules is of crucial importance to our approach. The extension of the set of heuristic rules, for instance by more smartly using the features of the partial timetable, seems to be an obvious direction for future work. Also, the sequential construction mechanism, which is fixed in its order, may not be optimal. Different orderings (a permutation of steps (i)–(iii) in Sec. 3.1) could be tested. We also plan to use the local search method developed in [6] in a Lamarckian learning approach to support our algorithm in finding good regions in the search space. We will test these different extensions to the algorithm in the near future.

Acknowledgments: This work was partly supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential program of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. D. Abramson. Constructing school timetables using simulated annealing: Sequential and parallel algorithms. *Management Science*, 37:98–113, 1991.
2. D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem. Technical report, 1991.
3. A. Colorni, M. Dorigo, and V. Maniezzo. Genetic algorithms and highly constrained problems. In *Parallel Problem Solving from Nature - Proceedings of the 1st Workshop*, volume PPSN 1, pages 55–59. Springer-Verlag, 1991.
4. U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, 22(1):25–40, 1995.
5. E. Hart, P.M. Ross, and J. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computing*, 6(1):61–80, 1998.
6. O. Rossi-Doria, C. Blum, J. Knowles, B. Paechter, M. Sampels, and K. Socha. A local search for the timetabling problem. To appear in the proceedings of PATAT'02, 2002.
7. A. Schaerf. Tabu search techniques for large high-school timetabling problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 363–368. AAAI Press/MIT Press, 1996.