Grouping Genetic Algorithm with Efficient Data Structures for the University Course Timetabling Problem

Felipe Arenales Santos · Alexandre C. B. Delbem

Keywords Grouping Genetic Algorithm · Timetabling Problem

1 Introduction

An efficient grouping genetic algorithm (GGA) was proposed by R. Lewis and B. Paechter (2007) to find feasible timetables for a version of the university course timetabling problem (UCTP). The algorithm proved to be efficient to construct feasible solutions for the UCTP benchmark instances (International Timetabling Competition 2002). In addition, the authors proposed a new set of harder instances, in order to better analyze their algorithm.

In this paper, we propose an adequate data structure for the algorithm and preprocessing techniques that signficantly improve its efficiency. The organization of the paper is as follows. Section 2 defines the UCTP and presents the main characteristics of the GGA. Section 3 and 4 describes our contributions for the GGA applied to UCTP. Finally, Section 5 presents computational experiments and the conclusions.

2 The University Course Timetabling Problem

A typical university timetabling problem consists of assigning a set $\mathbf{u}=\{u_1, ..., u_e\}$ of e events (classes, exams, lectures, etc.) to t timeslots and r rooms in such a way as to satisfy a set of constraints and to optimize an objective function (R. Lewis and B.

Felipe Arenales Santos
University of Sao Paulo
Sao Carlos, SP
Brazil
E-mail: fearenales@grad.icmc.usp.br

Alexandre Claudio Botazzo Delbem
University of Sao Paulo
Sao Carlos, SP
Brazil
E-mail: acbd@icmc.usp.br

Paechter 2007). The constraints are classified into two types (Burke et al. 1997): (i) hard constraints, which must be satisfied to a feasible timetable, and; (ii) soft constraints, which should be satisfied if possible. The hard constraints considered to this problem are:

1. Events requiring the same student should not be assigned to the same timeslot;
2. One room in one timeslot admits only one event;
3. All of the features required by an event should be satisfied by the room where the event is allocated, which has adequate capacity.

The problem is NP-Complete (Garey and Johnson 1979), and the number of possible assignments grows exponentially with the input data. As a consequence, exact methods for UCTP are not proper for large instances. Therefore, heuristic-based methods have been widely investigated in the literature (Costa 1994; Abramson, Krishnamoorthy and Dang 1996; Thompson and Dowsland 1998; Schaerf 1999; Paechter et al. 1998; Socha and Samples 2003).

Lewis and Paechter (2007) describe the applicability of a GGA to the UCTP, and propose a two-phase approach. In the first phase, the GGA improves the quality of the solutions (within a specified time) using a set of solution-builder heuristics, which speeds up the evolutionary process. In the second one, a local search algorithm does final improvements to promising solutions. However, as the input data increases too much (big cases of the proposed set), the convergence becomes slower and the first phase ends with high infeasibility level solutions. It should be worth noting that this paper deals with only the first phase and hard constraints.

3 Data Structures

Some of the data structs used by the GGA are trivial, such as the set **u**, implemented as an array, and the timetable, represented by the r$_{\mathbf{x}}$t matrix **T**, where the element t_{ij} is the event assigned to room i in timeslot j. Other matrices give the relation between students and events and between features and rooms. For further information, see Lewis and Paechter (2007).

The most complex data structure envolved in this GGA, called feasibility matrix **F**, stores the places (one place is one room in one timeslot) where the events can be feasibly allocated. Its most efficient implementation (by the processing aspect) consists of a matrix whose rows correspond to places, and columns to events. It can be done by initially converting a two-dimensional matrix (**G**, rooms by timeslots), that contains the feasibility information of a given event, to a one-dimensional array of places **H**. Figure 1 depicts this step.

$$\mathbf{G} = \begin{pmatrix} g_{11} & \cdots & g_{1t} \\ \vdots & \ddots & \vdots \\ g_{r1} & \cdots & g_{rt} \end{pmatrix} \rightarrow \mathbf{H} = \begin{pmatrix} g_{11} & \cdots & g_{1t} & \cdots & g_{r1} & \cdots & g_{rt} \end{pmatrix} = \begin{pmatrix} h_1 & \cdots & h_p \end{pmatrix}$$

Fig. 1: Linearization of the feasibility matrix to an event

Next, the resulting structure **H** is replicated e times (one for each event), resulting in the structure **F**. The **F** element $f_{ij} = 1$ if event j can be feasibly assigned to a place i, and 0 otherwise, i = 1,...,p and j = 1,...,e, see Figure 2.

$$\mathbf{F} = \begin{pmatrix} H_1^T & \cdots & H_e^T \end{pmatrix} = \begin{pmatrix} f_{11} & \cdots & f_{1e} \\ \vdots & \ddots & \vdots \\ f_{p1} & \cdots & f_{pe} \end{pmatrix}$$

Fig. 2: Feasibility matrix construction

In order to find out if one given event can be feasibly assigned to a place, no search is required, since the triple (*event*, *room*, *timeslot*) can be transformed into the pair (*event*, *place*) that corresponds to only one element of **F**. In this way, the cost to access this information is (in big-O notation) $O(1)$ (Cormen et al. 2001; Knuth 1998). It speeds up the generation of solutions, i.e. the GGA can performe a larger number of generations for the same period of time compared to other relatively higher cost data structures, such as dynamic lists and other non-static data structures. Moreover, it is important to highlight that the larger the number of generations, the better the quality of solutions at the end of the GGA.

4 Preprocessing Techniques

The values of the **F** elements are only related with the hard contraints satisfaction (see Section 2). Based on this, the hard constraints can be classified into two types: (i) static contraints, which define the assignment feasibility regardless the timetable state (i.e. the events allocated in it), as constraint 3; and (ii) dynamic constraints, which can change the feasibility of an assignment depending on the events already allocated in a timetable place, as constraints 1 and 2. Thus, the feasibility related to constraint 3 can be determined *a priori*. If an event requires a set of features and is required by n students, all the rooms that do not have all the required features or has the capacity less than n can not be feasibly assigned to this event anytime. This information is stored in matrix **A** (event by rooms) that lists the event-rooms conflicts. Element a_{ij} is 1 if the event i can not be assigned to the room j, i=1...e, j = 1,...,r; $a_{ij} = 0$ otherwise. Thus, the values of **F** to a new solution can be obtained directly from **A**, avoiding unneeded processing.

In addiction, the restriction number 1 can be modeled as a matrix **B** (event by event) that lists the event conflicts. The element $b_{ij} = 1$ if the event i requires a student also required by the event j, i = 1,...,e, j = 1,...,e, $i \neq j$; $b_{ij} = 0$ otherwise. Observe that every iteration of a solution building inserts one unallocated event into the solution being built.

5 Computational Results

The GGA and the proposed improvements was implemented in ANSI C under Linux (Ubuntu 8.04 distribution) on a computer with an Intel Pentium Dual-Core 1.73 GHz processor and 1 GB RAM. Considering the parameters as follow: timeslots number $t = 45$ (five days a week of five timeslots), mutation rate $m_r = 2$, inversion rate $i_r = 4$, recombination rate $rr = 1.0$, and population size $\rho = 50$. The input data is given by the instance sets. The experimental results was compared to the obtained by Lewis and Paechter (2007). The Figure 3 depicts the results obtained by the original implementation of the GGA and the implemented with improvements. The vertical axis corresponds to the distance to feasibility of the best solution present in the population. See Lewis and Paechter (2007) for further details about parameters and GGA implementation and run.

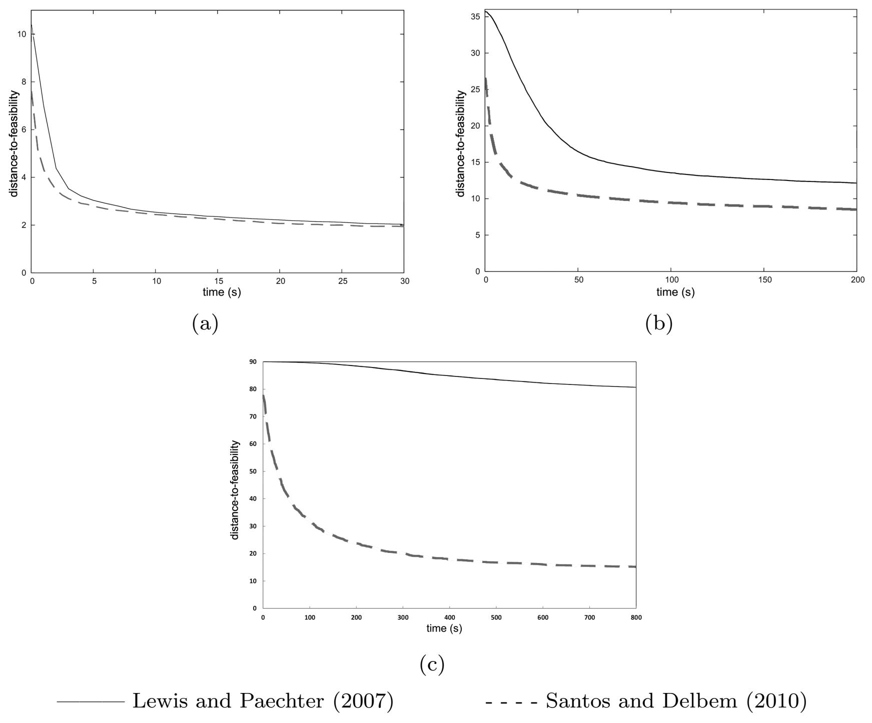

Fig. 3: Behavior of the algorithm to (a) small, (b) medium, and (c) large instances.

As can be noticed, the larger the input data, the greater the difference between the results obtained by the two implementations. It happens because as the input grows, the solution building time becomes significantly greater and the time saved is more expressive, showing the efficiency of the contributions proposed.

References

1. Abramson, D.; Krishnamoorthy, H.; Dang, H., Simulated Annealing Cooling Schedules for the School Timetabling Problem, *Asia-Pacific Journal of Operational Research*, vol. 16, pp. 1-22 (1996).
2. Burke, E. K.; Kingston, J.; Jackson, K.; et al., Automated university timetabling: the state of the art, *The Computer Journal*, vol. 40(9), pp. 565-571 (1997).
3. Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C., *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill (2001).
4. Costa, D., A tabu search algorithm for computing an operational timetable, *European Journal of Operational Research*, vol. 76, pp. 98-110 (1994).
5. Garey, M. R.; Johnson, D. S., *Computers and intractability: a guide to the theory of NP-completeness.* Freeman and Company, New York (1979).
6. Knuth, D., *The Art of Computer Programming*, Volume 1: *Fundamental Algorithms*, Third Edition. Addison-Wesley, pp. 107-123 (1997).
7. Lewis, R. and Paechter, B., Finding Feasible Timetables Using Group-Based Operators, *IEEE Transactions on Evolutionary Computation*, vol. 11(3), pp. 397-413 (2007).
8. Paechter, B.; Rankin, R.; Cumming A.; Fogarty, T., Timetabling the Classes of an Entire University with an Evolutionary Algorithm, in *Parallel Problem Solving from Nature* (PPSN) V (Lecture Notes in Computer Science, vol. 1498), Baeck T.; Eiben A.; Schoenauer M.; Schwefel, H., Eds. Berlin, Germany: Springer-Verlag, pp. 865-874 (1998).
9. Socha, K. and Samples, M., Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art, in *Evolutionary Computation in Combinatorial Optimization* (EVOCop) III (Lecture Notes in Computer Science vol. 2611), Dorigo, M.; Di Caro, G.; Sampels, M., Eds. Berlin, Germany: Springer-Verlag, pp. 334-345 (2003).
10. Thompson, J. M. and Dowsland, K. A., A Robust Simulated Annealing based Examination Timetabling System, *Computers and Operations Research*, vol. 25, pp. 637-648 (1998).