Evolutionary Timetabling Using Biased Genetic Operators

Daniel Danciu
CS Department, Babes-Bolyai University, Cluj Napoca, Romania

Evolutionary Algorithms have proved to be a flexible and effective technique for addressing various instances of the timetabling problem. This article describes an investigation and its results on an evolutionary approach to solving a particular class of highly constrained timetabling problems. The convergence speed of the evolution program has been significantly improved with the usage of biased operators, which generate offspring by preserving the building blocks of the parents. We also describe two metrics for measuring the efficiency of the genetic operators and how convergence speed has been improved by applying these metrics to fine-tune the probability of the genetic operators. Computational experiments over real test problems showed promising results.

Keywords: timetabling problem, biased operators, presence quotient, improvement rate.

1. Introduction

The timetabling problem is a particular optimization problem, and consists of arranging a sequence of meetings between teachers and students in a period of time, subject to constraints of several types. In this paper we consider a particular instance of the timetabling problem, called *school timetabling*, which consists of creating the daily schedule for all the classes of a high school, according to a predefined curriculum. Our main goal was to provide administrative staff of public high schools in Romania an alternative solution for the handmade timetables, which would both decrease the time spent in elaborating the timetable and produce better results. The particular characteristics observed for Romanian high schools are:

1) Each class of pupils is preassigned to a certain room. There are some exceptions to this rule, when a certain subject is to be held at a predetermined room or group of rooms (e.g. computer science is taught in labs, sports in the sports room etc.); in this case care should be taken that the number of subjects held simultaneously does not exceed the number of available rooms;

2) There are some subjects for which the class of pupils is divided into two subgroups, each subgroup being tutored by a different teacher (e.g. foreign languages);

3) Teaching starts at a predefined time for all classes and breaks are not allowed in the pupils' schedule. The number of hours per day varies between a predefined minimum and maximum;

4) Some of the subjects have to be grouped together in a continuous block.

The central problem in applying genetic algorithms to optimization problems is that of constraints - and the timetabling problem incorporates many nontrivial constraints of various kinds, which have been studied and categorized according to various criteria [5, 1]. Our approach of handling constraints is based on the principles of evolutionary programming [6]: adequate data structures and specialized genetic operators will take care that the potential solutions will satisfy the most important problem constraints. The remaining constraints are enforced via penalties, and have been grouped in the following categories:

— *teacher clashes*: the most prominent overall constraint (central to all timetabling problems) is that there should be no clashes [8]; that is, any pair of lectures which are supposed to share students or teachers should not be scheduled simultaneously;

— *organizational constraints* (e.g. having no more than two sports classes at the same time);

— *didactic constraints* (e.g. having more difficult subjects distributed in the first part of the day);

— *personal costs* (e.g. all teachers prefer to have their lectures for a day scheduled continuously, without breaks).

The convergence speed of the evolutionary algorithm is improved with the usage of biased genetic operators, which take into consideration information regarding violation of the penalties, calculated at the evaluation step. Biasing is applied to both the mutation and the crossover operators. The role of mutation in genetic algorithms using non-binary encodings of the solutions goes beyond recovering desirable genes that have been accidentally deleted from the population [9, 4]. In evolutionary programming, the mutation operator plays an important role as an exploratory tool as well, seeking to identify new desirable genetic structures. The biased mutation operators described in this paper are devised as an enhanced exploratory tool, which replaces "blind", random alterations of the chromosomes with changes that preserve building blocks and are more likely to produce better offspring. The role of crossover in genetic algorithms is to combine features in the mating parents to produce, hopefully, better offspring. Driven by this idea, there have been many efforts to improve the crossover operators for the timetabling problem, which often resulted in better, but more complex and computationally intensive operators [10, 11]. The biased crossover operator defined in section 2.4 has the advantage that it is both computationally efficient and clearly outperforms the unbiased version.

We measure the efficiency of the proposed genetic operators using two metrics: *improvement rate* and *presence quotient*. Even though defined in the timetabling context, the two metrics can be used to measure the effectiveness of the genetic operators of any evolutionary algorithm.

The paper is divided as follows: Section 2 describes the particular elements of the evolutionary algorithm that solves the high school timetabling problem: the solution encoding, the objective function, the genetic operators, and the initial population. Section 3 presents the operator efficiency metrics, the experimental setup and the results. Section 4 provides a concluding discussion and pinpoints some directions for further work.

2. The Problem

There are many versions of the timetabling problem, and most of the efficient approaches are tailored to a particular version. In our case, we have a list of teachers $T_1, T_2, \ldots, T_m$, a list of classes $C_1, C_2, \ldots, C_n$, a list of one hour time intervals $H_1, H_2, \ldots, H_p$ (partitioned into q subsets corresponding to the working days of the week), we have to find an optimal timetable, i.e. a timetable that satisfies the constraints described in Section 1.

2.1. Solution Encoding

The most natural (but, surprisingly, not used) data structure for the chromosome representation of a potential solution is the matrix representation in which the rows are the classes, the columns are the hours of the day, and the cells contain the teacher(s) lecturing a particular class at a given time, plus some additional information as described below.

Colorni et al. use in [1] a similar matrix representation, except that the rows in their case represent the list of teachers and the cells contain the classes. Gyori et al. [5] have introduced the notion of *set*, which consists of any number of teachers, classes and rooms, and allows class merging and splitting in a very flexible way, with the cost of imposing supplementary constraints (which were otherwise handled by the matrix representation) via penalties. In our approach, splitting of classes is handled by encoding supplementary information inside the cells of the matrix. An element x_{ij} of the chromosome matrix consists of a tuple (t_1, t_2, c, p). The first two components of the tuple contain the teachers that lecture C_i at time H_j (a value of 0 means “no teacher”). The third component, c, represents the category to which the subject belongs. Dividing subjects in categories is useful for enforcing supplementary constraints on certain subject groups. For example, we may require that for a particular subject group no more than two subjects belonging to that group can be taught simultaneously, due to some limited resources (computer science labs, sports rooms etc.). Since these rules are problem specific, they are imposed via the evaluation function. The forth component, p, encodes special properties of the lecture, which need to be considered by the genetic operators and/or the evaluation function:

— *Immutable*: The lecture cannot be moved by the genetic operator — it is required to remain as it was set up initially.

— *Block*: The lecture is part of a block (computer science labs are scheduled in blocks of two, three or four hours); genetic operators are allowed to move blocks, but are not allowed to split blocks.

— *Difficult*: The subject has an increased level of difficulty, and thus it would be desirable to schedule it at the beginning of the day; this property represents a didactic constraint and is handled via the evaluation function.

2.2. Objective Function

The representation of the solution described in the previous paragraph handles most of the important constraints. Thus, the initial population is generated so that the curriculum for each class is respected and there are no breaks in the students' schedule; genetic operators do not alter these constraints. The representation does not allow class clashes. The rest of the constraints are enforced via the selection pressure driven by the objective function.

The objective function measures a generalized cost, which represents the distance between a perfect timetable, respecting all constraints, including the organizational, didactic and personal requirements. Due to the contradictory nature of the constraints, in most of the cases the value of the objective function is positive for all timetable instances. The objective function for a timetable matrix X is defined as:

$$f(X) = \alpha \cdot c + \beta \cdot b + \gamma \cdot o(X) + \delta \cdot d(x) + \epsilon \cdot p(X)$$

where c is the number of clashes in X, b is the number of undesired breaks for teachers, $o(X)$ is the cost of unsatisfied organizational costs for X, $d(X)$ is the cost of unsatisfied didactic costs for X, and p is the cost of unsatisfied personal costs. The organizational, didactic and personal costs differ from one school to another, and thus o, d and p are defined ad-hoc for each particular instance. By choosing $\alpha \ll \beta \simeq \gamma \ll \delta \simeq \epsilon$, we induce a *hierarchical structure* in the objective function [1], so that we are able to drive the evolutionary process towards solving a particular category of constraints at each evolution stage.

The objective function represents the basis for the computation of the fitness function, which has the role of environmental feedback in the evolutionary process. The need to distinguish between the objective and the fitness function is given by the need of making the selection process more effective [6]. We have used $F(X) = 1/(1 + f(x))$ as the fitness function, combined with a *linear dynamic fitness scaling* [6, 1] procedure. Due to the nature of the objective function, it is essential to use the fitness scaling method, as otherwise the algorithm reduces to a blind search in the solution space.

2.3. Initial Population

Unlike other approaches [3, 1], which used the available handmade timetables as the initial population, we have used as a starting point a randomly generated population optimized using a simple heuristic approach, to reduce the number of clashes. We have also tested the algorithm with completely random populations; this made the evolutionary process somewhat longer, but did not significantly influence the final outcome. In both cases the initialization process was constructed so that all individuals respect the constraints that can be maintained via the solution encoding, as described in the previous paragraph.

2.4. Biased Operators

The genetic operators used in the algorithm, are the classical matrix genetic operators, as described in [1, 2]. We managed to improve the convergence speed of the algorithm by altering the above operators to act in a biased way instead of a uniform way. The idea is that both the crossover and the mutation operators randomly pick some lines (classes) and columns (time intervals) that are subject to their action. Whereas in most evolutionary approaches the nature of the solution encoding does not allow localizing the genes that are the cause of the constraint violations, in our matrix representation this is quite straightforward to do.

When evaluating an individual, the evaluation algorithm counts, for each row and each column, the number of cells that are causing constraint violations (clashes, breaks in the teachers' schedule etc.) and calculates a weighted sum of these infeasibilities, in a similar manner as for the hierarchical objective function. This sum represents a violation score that can be used to infer which classes and time intervals are more problematic in the timetable represented by the current individual. We put this information to use in two ways: direct the mutation operators towards problematic genes and preserve building blocks in crossover operators.

Most of the matrix mutation operators basically reduce to exchanging two groups of cells, randomly selected. We can use the violation score for each row and column to direct the action of the mutation operators in a very simple way. The idea is to move away the cells that are causing constraint violations into zones without or with less constraint violations. Thus, when selecting the first group of cells, we assign each row and column a probability of selection, which is directly proportional to the violation score. We can then use any of the standard evolutionary selection schemes to select the rows and columns that belong to the first group of cells. We have chosen to use the straightforward roulette wheel method for this. A good reason for this choice is simplicity. For example, we can easily select a column by generating a random number between 0 and the total violation score for all columns. After that, we traverse the columns, decreasing the random number with the column's violation score, until it becomes negative. The column where we stopped is selected for mutation. Rows can be selected in a similar manner.

The idea of using violation-directed mutation operators is not new. Ross et al. [7], have devised and studied an improved mutation operator, called *directed mutation*, which exchanges two genes selected using constraint violation information in a similar way. The differences are that Ross et al. also consider constraint violation information for choosing the allele where the first group is going to be placed, whereas in our case the second group of cells is randomly selected, and Ross et al. only consider single gene mutations, while the matrix encoding used in this paper naturally allows biasing for all mutation operators (for example, day-swap mutation, which exchanges two blocks of genes). The reason for randomly selecting the second group of cells is that calculating the constraint violation score for each possible allele position is computationally expensive and seems more adequate for a genetic repair algorithm ([2, 1]) than for a mutation operator.

The violation score can also be used for devising efficient crossover operators, which tend to preserve building blocks. It is known that crossover is the main source of exploration in the evolutionary process. The problem in crossover is to find the proper tradeoff between preservation of schemata (and thus, of the building blocks) and the effective recombination. In other words, we need to find a balance between the extent to which good solutions obtained at a certain phase are preserved and the extent to which new regions of the search space are explored. By biasing the crossover operator we address exactly this problem.

Our algorithm uses two crossover operators: *section crossover* and *line-swap crossover*. Section crossover takes two parents and generates two offspring by horizontally cutting the parent matrices (at a random line) and exchanging the lines below the cut. Similarly, line-swap crossover takes two parents and generates two offspring by selecting several lines from each parent that are going to be replaced with lines from the other parent. Line-swap crossover can be easily enhanced by making use of the violation score. Instead of randomly selecting the lines that are going to be swapped, we bias the probability of selection, so that lines having a larger violation score have more chances to be selected for being replaced. Thus, the resulting operator will tend to keep the building blocks (solved subproblems) of the parents and pass it further to the offspring. The usage of biased operators has brought considerable improvements in the efficiency of the genetic operators and thus in the evolutionary process itself, as described in the next section.

3. Efficiency Measurement of the Genetic Operators

Given the matrix representation of the timetable, it is very easy to devise several mutation and crossover operators [2]: cell-swap mutation, day-swap mutation, day-permutation mutation, column-swap mutation, section crossover, line crossover etc.

The list could go on with several other operators, but the question is how much does each of these operators improve the convergence speed of the evolutionary algorithm? One could experiment with assigning different probabilities to each of the operators and see how the quality of the results is affected over several test runs. However, this method does not always give conclusive results and does not offer concrete comparison criteria between the genetic operators. We have devised two simple metrics which characterize the efficiency of the genetic operators: *improvement rate* and *presence quotient*.

The *improvement rate* is defined as the number of times the operator actually brings an improvement on the individual divided by the total number of times it is applied. The *presence quotient*'s calculation is slightly more complex. For each individual in the population, we store the sequence of operators that has lead to its generation. We call this sequence *operator chain*. Then, we count how many times a specified operator appears in the operator chain and divide this number with the expected number of appearances, obtaining the *presence quotient*. If operators would be equally efficient, the *presence quotient* should be approximately equal to 1. However, practice has proven that this is not the case. Naturally, we are interested in the presence quotient of the best individuals in the population, as this gives a good hint upon how much each operator has contributed to obtaining the solution of the problem.

In order to test the performance of our evolution program, we implemented it in Java (source code available at `http://www.danciu.ro/tt`), and tested it using concrete inputs. The input data contained $m = 54$ teachers and $n = 23$ classes. We considered $q = 5$ working days, each day having at most 7 hours; the number of time intervals is $p = 35$. The total number of hours to be taught is 739, out of which 123 are taught in subgroups (two teachers at the same time), which gives an average of 16 hours per teacher and a coverage of 91% of the available time. The generated timetables were better in terms of cost than the handmade ones: they contained no clashes, no undesired breaks in the teachers' schedules and managed to fulfill most of the organizational, didactic and personal goals.

Table 1 summarizes the average values for the improvement rate and presence quotient of the six genetic operators enumerated above plus the biased versions of cell-swap mutation and line-swap crossover operators. The superiority of the biased operators over their unbiased counterparts is clearly proved by both a higher improvement rate and a higher presence quotient.

Genetic operator	Impr. rate	Presence quotient
Section crossover	0.14	1.30
Line-swap crossover	0.09	1.07
Biased line-swap crossover	0.12	1.30
Cell-swap mutation	0.03	0.95
Biased cell-swap mutation	0.05	1.24
Day-permutation mutation	0.02	1.58
Day-swap mutation	0.01	0.21
Column-swap mutation	0.007	0.07

Table 1. Average values of the efficiency metrics for the genetic operators over ten test runs.

By analyzing the values in Table 1 we came to the following conclusions:

— the biased versions of the operators are clearly superior to their unbiased counterparts, thus the unbiased versions can be eliminated;

— the cell-swap mutation operator is far more efficient than the other mutation operators and, from this point of view, it competes with the crossover operators. Thus, increasing its probability of application will have the benefit of both enhancing population diversity and increasing convergence speed. This is proven by the graph in Fig. 1, which plots the values of the best individual over each generation, for two different probability configurations of the evolution program: the initial configuration and the fine-tuned configuration, based on the operator metrics. The result is surprising from the classical genetic algorithms theoretical point view: an algorithm with a mutation rate higher than the crossover rate clearly outperformed the classical algorithm with low mutation rates;

— day-swap and column-swap mutation operators bring little contribution to the convergence speed and their probability can be reduced;

— since the values of the improvement rate and of the presence quotient are consistent (lower improvement rate corresponds to lower presence quotient and vice versa) we can conclude that the two metrics are a valuable tool for empirically measuring the relative efficiency of the genetic operators.

It is interesting to note that higher values of the improvement rate for the operators over two distinct configurations of the evolutionary algorithm do not necessarily imply a better convergence rate. On the contrary, by introducing inefficient genetic operators, which tend to slow down the evolutionary process, the improvement rates for the rest of the operators get higher, but this is only because the new operators increase the diversity of the population and thus give more chances for the other operators to repair below-average individuals.

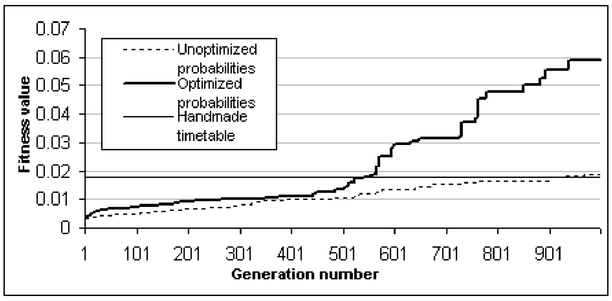

Fig. 1. Best individual fitness over 1000 generations for the initial and the fine-tuned probabilities; repeated runs give very similar results.

4. Conclusion

We have shown that concrete timetabling problems can be effectively addressed by an evolutionary algorithm which uses a direct matrix representation and well-selected genetic operators. The biased genetic operators have been proposed as an alternative to the classic matrix operators. The operator efficiency metrics measure the effectiveness of the genetic operators from two points of view. The presence quotient shows how much each operator has contributed to creating the best individual, while the improvement rate measures how much each operator has contributed to improving or repairing the individuals in the population. The experiments performed have shown that these two metrics are consistent and can be reliably used to compare the relative efficiency of the genetic operators defined for an evolution program. The proposed algorithm will be experimentally used in the next year at the Computer Science high school in Braşov.

References

[1] Colorni A., Dorigo M., Maniezzo V., Metaheuristics For High-School Timetabling, *Computational Optimization and Applications* 1997; 9(3): pp. 277–298.

[2] Danciu D., Sasu L.M., Solving The Timetable Problem With Multiple Constraints Using Genetic Algorithms, in: Orman G., editor, *Proceedings of the 2nd International Conference on Symmetry and Antisymmetry*; 2000 Jun 29–30; Brasov, Romania. Transilvania University, Brasov; 2000., pp. 137–145.

[3] Filho R.G., Lorena L.A.N., A Constructive Evolutionary Approach to School Timetabling, in: Boers EJW et al. editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops*, 2001, Apr 18–20; Como, Italy, 2001, pp. 130–139.

[4] Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, MA: Addison-Wesley; 1989.

[5] Gyori S., Petres Z., Varkonyi-Koczy A.R., Genetic Algorithms in Timetabling. A New Approach, *MFT Periodika* 2001. www.mft.hu/hallg/200107.pdf [02/15/2003].

[6] Michalewicz Z., *Genetic Algorithms + Data Structures = Evolution Programs*, New York: Springer-Verlag; 1994.

[7] Ross P.M., Corne D., Fang H.L., Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation, in: Davidor Y., Schwefel H.P., Manner R., editors, *Proceedings of PPSN III*; 1994 Oct 9–14; Jerusalem, Israel, pp. 556–565.

[8] Ross P.M., Corne D., Fang H.L., Successful Lecture Timetabling with Evolutionary Algorithms, in: Eiben A.E., Manderick B. and Ruttkay Zs., editors, *Proceedings of the ECAI 94 Workshop on Applications of EAs*; 1994 Aug 9; Amsterdam, Netherlands. ECCAI; 1995, pp. 23–34.

[9] Tate D.M., Smith A.E., Expected allele coverage and the role of mutation, in: Forrest S., editor, *Proceedings of the Fifth International Conference on GAs*; 1993 July 17–21; Urbana, IL. Urbana-Champaign: University of Illinois; 1993, pp. 31–37.

[10] TERASHIMA-MARIN H., ROSS P.M., VALENZUELA-RENDON M., Clique-Based Crossover For Solving the Timetabling Problem with GAs, in: Schoenauer M. et al., editors, *Proceedings of CEC 99 Conference*; 1999 Jul 6–9; Washington, USA, pp. 1200–1206.

[11] YAMADA T., NAKANO R., Scheduling by Genetic Local Search with Multi-Step Crossover, in: Voigt H-M. et al., editors, *Proceedings of PPSN IV*; 1996 Sep 22–26; Berlin, Germany, pp. 960–969.

Received: June, 2003
Accepted: September, 2003

Contact address:
Daniel Danciu
CS Department
Babes-Bolyai University
M. Kogălniceanu 1
RO-3400 Cluj Napoca
Romania
e-mail: `daniel@danciu.ro`

DANIEL DANCIU is presently an independent consultant and is involved in the development of several industry projects. At the same time he is doing research for his PhD thesis concerning the application of evolutionary techniques to various constrained optimization problems. As a consultant, he specializes in the design and implementation of performance monitoring tools. He completed his scientific training at the universities in Brasov and Cluj-Napoca and worked for a number of years at Transilvania University of Brasov.