A Genetic Algorithm for a Student Timetabling Problem

Milena N. Karova[1] Violeta T. Bojikova[2] Radoslav E. Mladenov[3]

Abstract - **This paper introduces a flexible method for scheduling a timetable using a genetic algorithm. The timetabling problem comes up in every educational institution. It is a special kind of optimization problem. A timetable is explained as a schedule with constraints placed upon it. There had been many attempts to address this problem using classical methods, such as integer programming and graph theory algorithms without much success. These methods also are difficult to automate the process. The solution, which this paper offers, includes a genetic algorithm implementation in order to give a maximal approximation of the problem, modifying a generated solution with genetic operators.**

Keywords: **genetic algorithm, timetabling, scheduling, constraints, optimization, approximation, selection, genetic operator**

I. INTRODUCTION

The timetabling problems belong to the NP-hard problems class and it is quite difficult to solve them with conventional methods including iterative and recursive algorithms. These problems are generally characterized as constraint satisfaction problems. That's why we use two general categories of constraints – hard constraints and soft constraints. The basic objective in solving the problem is to allocate events to time slots while minimizing constraint violation. This approach needs robust heuristics which can evolve and evaluate the set of solution candidates called chromosomes. The scheduling doesn't mean just arranging events in the time slots but also observing some additional rules – teachers may be busy in certain weekdays, some events may require a specific time slot and so on.

II. CONSTRAINTS

The genetic algorithm which we use, observes two generic groups of constraints – soft constraints and hard constraints. Hard constraints are constraints which mustn't be broken in order to have a regular timetable. These include:

- a teacher giving two or more lessons at the same time
- a group attending two or more classes at the same time
- a room being occupied by two or more groups at the same time

[1]Milena N. Karova is with the department of Computer Science, Studentska 1, Technical University Varna Email: mkarova@ieee.bg
[2]Violeta T. Bojikova is with the department of Computer Science, Studentska 1, Technical University Varna Email:vbojikova2000@yahoo.com
[3]Radoslav Mladenov is student, department of Computer Science, Technical University Varna Email: radoslavmladenov@abv.bg

Such constraints include more sophisticated problems to be solved – room capacity problems, time violations etc. Hard constraints have to be taken into consideration very strictly because the timetables that violate just one of them are practically unusable. Soft constraints are not so important but they mustn't be belittled. They involve restrictions as:

- reducing the void time slots
- setting the lectures before seminaries
- selecting preferred time slots

Soft constraints offer more gentle options for constructing the timetable. Soft constraints are the tools for customizing a particular timetable without many efforts. The soft constraints' violations will not make a timetable unusable; it will only be more discursive. A comparison function is necessary, to define when a chromosome is better than another chromosome. Comparing two chromosomes (and thus obtaining the best individual) consists in choosing the individual with the best hard fitness (in case of ties, choosing the individual with the best soft fitness). Note that the best fitness means the lowest fitness factor (the fitness factor is decreasing with better individuals) and thus maybe a more appropriate name for this fitness factor would have been "conflict factor".

III. EVENTS AND SOLUTIONS

Every single solution generated by the algorithm is represented by a chromosome. The chromosome itself is an indivisible unit which forms the total amount of solutions called generation. Each chromosome is made up of a set of genes (the smallest information carrying unit of a chromosome). In our approach, inside the chromosome, there is a gene for each activity in the timetable. This gene represents the scheduled time of the corresponding event. So, a chromosome is actually an array of genes, each gene representing the starting day and hour of an event.

The chromosomes are built with direct encoding. This means that the chromosome is not a sequence of logical 0 and 1 but contains more meaningful information. The single most important attribute of the chromosome is its length. The length is determined from the total amount of events, taking part in the schedule. The event is a discrete container which includes a teacher, a group and a room. Each gene in the chromosomes represents the time slot for the corresponding event in the set of events. To reduce the space representation of the whole solution, we linearize the chromosome. We use 6 fixed time slots per day, so representing a single time slot we should need a matrix. Linearization removes the

indispensability of using a matrix. If there are 5 working days the matrix will be 6x5, but if we linearize, we will need only a vector with 30 fields. Then the generation would be a matrix (not a cube) and the total number of solution candidates would be a cube (not a 4-dimensional hypercube).

IV. GENETIC OPERATORS

The genetic algorithm we present uses three genetic operators to perform the evolution process. They are crossover, mutation and preserving. The crossover is a biological term which is widely used in the genetic algorithms theory. It is the phase in our algorithm where we produce new solution candidates from existing ones. Crossover is committed in two processes: one is the parent selection, where we select two chromosomes by following some criteria. In this process we try to give a better solution using old solutions. After the two parent chromosomes had been selected, we need to determine how to combine the two chromosomes to produce new better solutions. There are two common methods of doing so – single-point crossover or double-point crossover where we segment the chromosomes randomly and exchange their genetic information. We also offer another crossover variation – crossover with elitism. Elitism is the mechanism of evaluating and finding the best chromosome from certain population. We use a single-point crossover which consists of finding a point of division (it's randomly chosen) and then swapping the genetic information between the two adult chromosomes. The second genetic operator which is used in the algorithm is the mutation operator. It introduces a rapid change in the genetic material in a single chromosome. This operator must be used rarely because it brings unforeseen changes which are likely to deteriorate the chromosome's fitness value. Sometimes mutation surprisingly gives enormous improvements in the solution's overall structure. Mutation ratio should be kept small, often 5% or less because it is an irreversible process which may worsen the solution rather than improve it. The last genetic operator we use is the preserving. It is just copying one solution form the old population into the new generation.

V. SELECTION AND EVOLUTION

We offer two methods of selection. These are the three-tournament selection and semi-elitism method. The three-tournament selection involves selecting three absolutely random chromosomes from the current generation. They are evaluated and then sorted ascending. The worst of the three is eliminated from consequent evolution. In the crossover phase are used the better two chromosomes and in mutation and preserving is used only the best of the three randomly chosen chromosomes. The semi-elitism selection method is a variant of the classical elitism. It finds the best chromosome but does not find the next better chromosome. Instead it chooses randomly the second genetic operand. In mutation and preservation only the best chromosome of the whole population takes place. We do not use the best two chromosomes in crossover due to the aggregate violations avoiding. It is proven that in most cases the first two best chromosomes are almost identical and it is likely that swapping them will give no good result. That's why we do not use them.

Selection is not enough to retrieve the new solution candidates. Evolution is the process of choosing which genetic operator to execute. There are probabilities for each genetic operator to switch. The program allows the user to adjust these values according to the specific situation. We assign a probabilities area which is units 100 long (if we work in per cent). Consider the following situation: crossover probability: 85%, mutation probability: 5%, preserve: 10%. The probabilities area will be divided into three fields: (0,5], (5,90] and (90,100]. Then a random number between 1 and 100 is selected and according to the probabilities area, the genetic operator is chosen. The next step in the implementation of the genetic algorithm is the evaluation.

VI. EVALUATION

Evaluating the chromosomes is carried out in two phases – evaluating the hard fitness and the soft fitness of a particular chromosome. Hard constraints must not be violated in order to have a useable timetable, so the primary criterion in the evaluation is the hard fitness value. It is formed as a simple sum of penalty points. A penalty point is given if a certain hard constraint isn't obeyed. The total hard fitness is evaluated by the following formulae Eq. (1):

$$TotalHardFitness = \sum_{k=0}^{N} fi_k \qquad (1)$$

The total hard fitness helps as at evaluating the algorithm's overall efficiency. It is used in graphics, visualizing the sequence of population propagation. Each chromosome is evaluated with arithmetic sum of its individual time slot violations. This process looks like Eq. (2):

$$Fitness = \sum_{k=0}^{n} TimeSlotClash_k \qquad (2)$$

n is the total amount of events taking part in the timetable construction. This is very important because just one violation in time slots makes the schedule unusable without manual corrections, but the purpose of our algorithm is to avoid manual corrections of a generated solution. The soft constraints evaluation is the same, but when comparing two chromosomes the hard fitness factor is more important than the soft fitness factor. The purpose of the genetic algorithm at all is to fulfill the hard constraints and to reduce the soft constraints maximally.

VII. EXPERIMENTAL DATA

An experimental implementation was done as a C++ object oriented program. It directly uses the evolution data for the particular timetable and draws a graphic which shows how the

conflicts go down. The black line represents the hard constraints and the red line represents the soft fitness. Here we show how the population size affects the speed of finding a satisfactory solution. The following table presents the experiments Table I:

TABLEI

Population size	***Selection***
32	Tournament
32	Semi-elitism
128	Tournament
128	Semi-elitism

First, we experiment with a population of 32 solution candidates using tournament selection. The result is shown in the following figure Fig. 1:

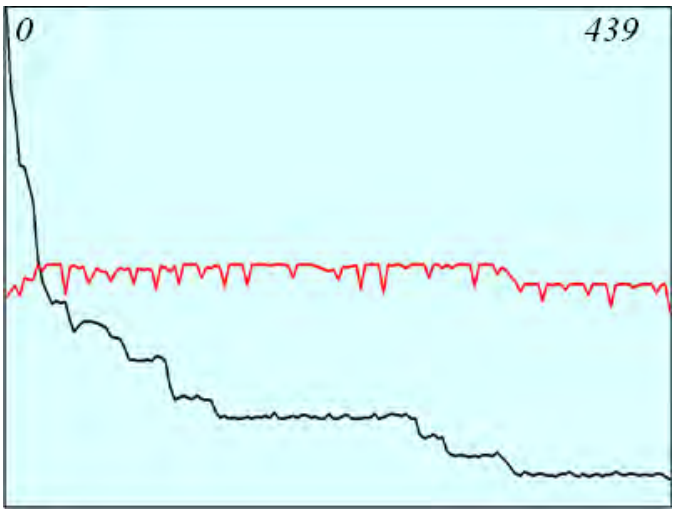

Fig. 1

The population number 439 contains a chromosome which is fulfilled every hard constraint in its set. The next figure shows the results of the same population size, but using elitism Fig. 2:

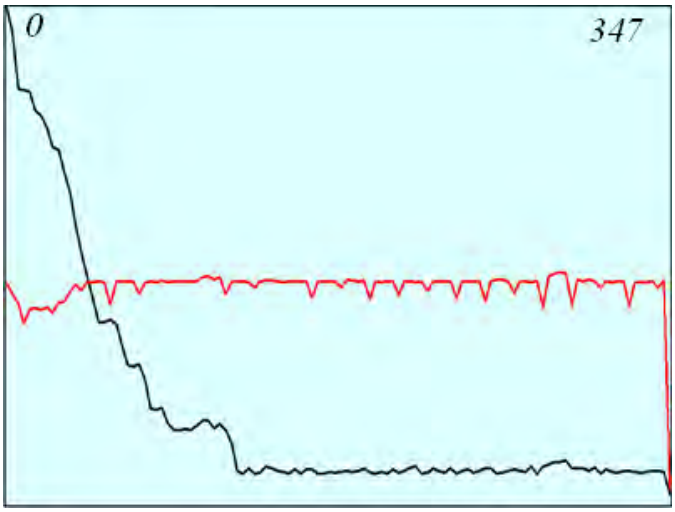

Fig. 2

It is obvious that using elitism reduces the total number of generations which are evolved. Also the better solution is found faster than the tournament selection. The population size here is more important than the selection method. The following two figures show how the solution improves faster and faster when the population size is 128. The time taken however increases as the population size increases, so when dealing with larger data sets it's necessary to choose the most appropriate population size. Fig. 3 shows that when the population size is bigger, the number of generations is lower. It's using a tournament selection.

Fig. 3

The fourth experiment is the same as the third experiment except for the selection method. Here is used elitism which reduces the generations Fig. 4.

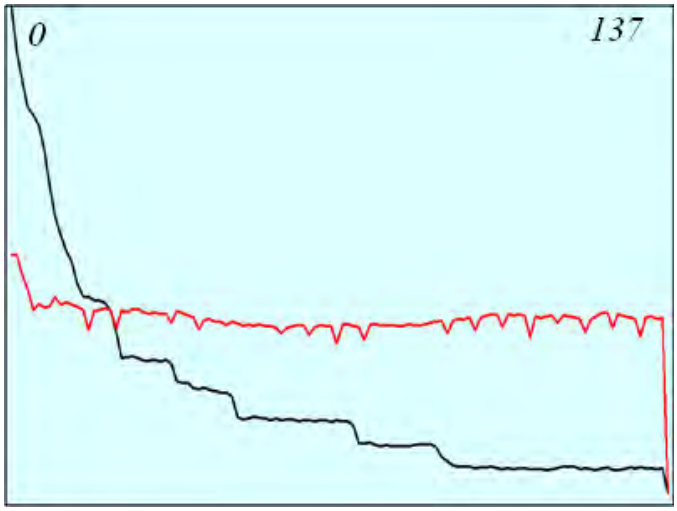

Fig. 4

All of the above experiments use these genetic operator probabilities:

Crossover – 87%

Mutation – 5%
Preserving – 8%

The program allows these values to be adjusted whenever the user wants to do so. For the bulk of data sets the above values give pretty good results and that's why we used them in the experiments.

VIII. CONCLUSION

We presented a algorithm for solving timetabling problems, which combines principles of the local search with other techniques for solving constraint satisfaction problems. The proposed principles can be applied to other constraints satisfaction problems; they can be easily extended to cover additional hard and soft constraints.

Further research is oriented both theoretically, to formalize the techniques and to put them in a wider context of constraint programming, and practically, to implement the above described genetic algorithm.

REFERENCES

[1] Abramson D., Abela, A Parallel genetic algorithm for Solving the school Timetabling Problem., Royal Melbourne Institute of technology, 1991

[2] Burke E., Ross P., Practice and Theory of Automated Timetabling. Lectures Notes In Computer Science Springer, Berlin 1996

[3] Michalewicz Z., Janikow C., Handling constraints in genetic algorithms. In Proceeding of the 4th International Conference in Gas. Morgan Kauffman, 1991.

[4] Michalewicz Z, Genetic Algorithms+ Data Structures = Evolution Programs, Springer Verlag, 1992.

[5] Syswerda G. Uniform crossover in genetic algorithms. Proceeding of Third International Conference of Genetic algorithms

[6] Corne D., Ogden J., Evolutionary Optimisation of Methodist Preaching Timetables, Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II, p.142-155, August 20-22 1997.

[7] Goldberg D., Web courses, http://www.engr.uiuc.edu/OCEE, 2000