

# GENETSKI ALGORITAM SA OPTIMIZACIONIM METODAMA U REŠAVANJU PROBLEMA RASPOREDA ČASOVA

## GENETIC ALGORITHM WITH OPTIMIZATION METHODS IN SCHOOL TIMETABLING PROBLEMS

Edin Mulalić, Leonid Stoimenov, Branislav Randelović  
*Faculty of Electronic Engineering, University of Niš*

**Sadržaj** – Problem kreiranja rasporeda jeste primer kompleksnog optimizacionog problema. Tradicionalni algoritmi pretraživanja nisu naročito efikasni u resavanju ovog problema. Ovaj rad opisuje upotrebu genetskog algoritma u procesu nalazenja “korektnog rasporeda” za obrazovnu ustanovu kao i neke optimizacione metode koje znacajno ubrzavaju algoritam. Metode predložene od strane drugih autora su kratko pomenute, dok su originalna rešenja objašnjenja sa više detalja. Predložene tehnike su testirane na realnim primerima i, na kraju rada, prikazani su eksperimentalni rezultati.

**Abstract** - Timetabling problem is an example of complex optimization problem. Traditional search algorithms are not very effective in solving this problem. This paper describes usage of genetic algorithm in process of finding a “correct timetable” for an educational institution, as well as some of the optimization methods which can significantly speed up algorithm. Some optimization methods proposed by other authors are briefly mentioned, while original improvements are explained with more details. Proposed techniques are tested on real life data and, at the end of the paper, some experimental results are presented.

### 1. INTRODUCTION

Timetabling problem belongs to the wide group of complex optimization problems which have too big search space to be solved using traditional approach. It can be found in real life situation – in schools, banks, airports etc. Actually, whenever exists need for planning any kind of resources and their availability in time, it can be classified as timetabling problem. It is possible to investigate timetabling problem in general. But specific environments have specific constraints, thus, analyzing specific problem can help in achieving better performances. This paper is focused on problem of creating school timetables, where classrooms, teachers and student groups are considered to be resources.

Traditionally, timetabling problem in educational institutions has always been solved by humans. But even an experienced person needs plenty of time to finish the job.

And if some of preconditions are changed, the whole timetable becomes unusable and the entire process has to be restarted. Thus, computer software capable of creating a good timetable would be very useful.

This paper is organized as follows. Section 2 describes basics of school timetabling problem. Section 3 provides introduction to genetic algorithms. Sections 4 to 9 describe components of genetic algorithm and explain usage of those components in implementation. In Section 10, results of our experimental work are given. Finally, in Section 11 our discussion is concluded.

### 2. THE SCHOOL TIMETABLING PROBLEM

Timetabling problem is an example of complex optimization problem where the traditional search algorithms are not very effective in solving this problem. Usage of genetic algorithm in process of finding a “correct timetable” for an educational institution is one solution of this problem.

Given sets of:

1. classrooms  $R = \{r_1, r_2, \dots, r_n\}$ ,
2. teachers  $T = \{t_1, t_2, \dots, t_m\}$ ,
3. classes  $C = \{c_1, c_2, \dots, c_k\}$ ,
4. hard constraints  $H = \{h_1, h_2, \dots, h_p\}$ ,
5. soft constraints  $S = \{s_1, s_2, \dots, s_q\}$ ,

it is required to construct a correct timetable. The phrase “correct timetable” needs some explanation. It is difficult to label a certain solution as a “good solution”. It is even more difficult to measure how good it is. Much easier is to recognize which timetable is unusable. So, by “correct timetable”, is meant timetable that satisfies all hard and as many as possible soft constraints.

Set of constraints consists of hard constraints (H) and soft constraints (S). Constraints define values that the problem variables can take simultaneously [1]. Hard constraints are preconditions that have to be satisfied for timetable to be

useful. Violating just one hard constraint means that the whole timetable has to be discarded. Examples of hard constraints [2]:

1. a teacher has to be in two different classrooms at the same time,
2. two different classes have to be in the same classroom at the same time (merged classes do not cause conflict),
3. teacher availability during semester etc.

Additional hard constraints can be defined considering specific demands for concrete educational institution.

Soft constraints are preconditions which don't have to be necessary satisfied to obtain a useful timetable. Those preconditions can be in conflict among themselves, thus, it may not even be possible to satisfy all of them. Those constraints can be associated with priorities, to distinguish which are more important to be satisfied. Which soft constraints would be defined, depends mostly on concrete demands from institution, but some examples can be [2]:

1. clustering lessons - holding two or three lessons of the same subject consecutively,
2. the break between lessons should be minimized for students and for teachers,
3. the preferences of teachers,
4. equal (or near equal) number of lessons per day etc.

It is important to say that the same constraint can be labeled as "soft" for one educational institution and as "hard" for another. Thus, it is not possible to define constraints without knowing specific demands from educational institution.

In attempt to create a correct timetable, one can apply direct search. Combination of heuristic and some sort of backtracking can give result. But, creating timetable which satisfies all hard constraints and maximal number of soft constraints, like most optimization problems, is computationally NP-hard [3]. Problem's complexity can create search space which is too big to be explored by direct search within reasonable time. Genetic algorithm (GA) can be used to surpass these problems [4]. Genetic algorithm is not an algorithm in true meaning of that word. Providing only basic directions in solving a problem, it is more a way to approach the problem than precisely defined set of instructions. It is possible to represent data in many different ways. Also, there are many ways to define and realize each of genetic operators. Which way is the best is an open question. Therefore, the main challenge in creating a software using GA is not to apply GA, but to define and implement all of its parts in the best possible way in order to achieve maximal performances of software.

### 3. GENETIC ALGORITHM FOR SCHOOL TIMETABLING PROBLEM

Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary

biology. GA is usually used as a search method in big and complex search space. It belongs into local search method, therefore it is an incomplete search. Basic concepts from biology taken by GA are *individual*, *population*, *reproduction* (*crossover*, *recombination*), *selection*, *inheritance*, *mutation* and *fitness*.

An *individual* is a potential solution. In timetabling problem, an individual is a timetable which may or may not satisfy all constraints. For each individual, *fitness function* tells how "good" that potential solution is (or how adapted to the environment it is). Fitness function is always problem dependent. Selection of fitness function can affect speed of convergence to the correct solution. Group of individuals makes a *population*. Creating a new individual from the existing population is called reproduction. Reproduction helps in creating new potential solutions which would *inherit* good characteristic of explored potential solutions, but also try to explore more of "unknown territory". Which individuals will reproduce is defined by *selection* process. Just like in real life situations, the individuals which are better adapted to the environment have more chances to reproduce. *Mutations* exist to prevent individuals to become too similar and to avoid local extreme values of fitness function. Mutations bring new quality to the population by giving individuals with characteristics that can not be inherited from existing individuals. That speeds up process of convergence to the correct solution. Creating new individuals is repeated until *terminating conditions* have been reached. Terminating conditions are also problem dependent. The most common are:

1. first correct solution has been found
2. overall fitness function stopped decreasing
3. fixed number of generation reached
4. too much time elapsed
5. "good enough" solution has been found
6. combination of the above.

Pseudo code for GA can look like this:

```

generate (random) initial population p;
repeat
    evaluate fitness function for each
    member of the population p;
    select pairs for reproduction (from
    population p) using their fitness
    values;
    breed new generation q from p using
    crossover and mutations;
    p ← q;
until terminating conditions;

```

#### 3.1 Representation of individual

Each individual presents one possible solution. There are many ways to represent one timetable, but used structure has to have following properties:

1. it has to contain complete information about teachers, classes, lessons and classrooms
2. it has to be possible to efficiently traverse the structure in order to check existence of constraint violations
3. it has to support efficient constructing and exchange of elements because of genetic operations crossover and mutations.

In our implementation, basic data unit is called “slot”. Slot structure is shown in Figure 1.

<ul style="list-style-type: none"> <li>• Teacher</li> <li>• Subject</li> <li>• Classroom</li> <li>• Serial number</li> </ul>
--

Figure 1. Slot structure

Timetable which describes one student group is composed of many slots, like in Figure 2. Each slot in one group, including empty slots (those without teacher, subject and classroom), has a unique serial number which is used to distinguish that slot from others in crossover operation.

Class <sub>i</sub>					
	MON	TUE	WED	THU	FRI
1.	Slot <sub>1</sub>	Slot <sub>2</sub>	...		
2.					
3.		Slot <sub>k</sub>			
4.					
...	...	...	...	...	...
mnlp					

Figure 2. Part of timetable representing one student group

The whole individual is array of group timetables and can be represented as a matrix with *max\_number\_of\_lessons\_per\_day (mnlp)* rows, and *5 \* number\_of\_classes (5\*nc)* columns (Figure 3).

	Class <sub>1</sub>	Class <sub>2</sub>	...	Class <sub>nc</sub>
1.				
2.				
...				
mnlp				

Figure 3. Full timetable (one individual)

### 3.2 Initialization

Creating first population goes individual by individual. Each individual is generated group by group, and each group is generated slot by slot. Two slots which have the

same values for teacher, classroom and subject and are associated with the same group in different individuals, have the same serial number. In pure GA, initial set of individuals is usually generated randomly. But if we do not generate initial population completely randomly, we can have huge advantage in start. First of all, some of preferred characteristics could have been incorporated in all of initial individuals assuring that their offspring inherits those characteristics. Secondly, during generating individuals, we can check if any of hard constraints are violated and (if possible) try to avoid those.

### 3.3 Fitness Function

Fitness function can be evaluated using following formulas:

$$f = h + \gamma * s / m \quad (1)$$

$$h = \sum_{i=1}^p \alpha_i * h_i \quad (2)$$

$$s = \sum_{j=1}^q \beta_j * s_j \quad (3)$$

where  $\gamma = 0$  while  $h > 0$  and  $\gamma = 1$  when  $h = 0$ ,  $m$  is upper bound of tolerated violations of soft constraints.  $h_i$  and  $s_j$  are number of conflicts caused by unsatisfying  $i$ -th hard constraint and  $j$ -th soft constraint, respectively.  $\alpha_i$  and  $\beta_j$  are used to define priorities among constraints. The main goal is to find individual with  $\min(f)$ . Useful timetable has to satisfy  $h = 0$ . It is possible to use more complex fitness functions, like in [2] and [5]. More complex fitness function may lead to faster convergence in term of number of generations, but needs more time to be evaluated.

### 3.4. Selection

During the “mating” process, individuals which participate in reproduction are selected according to their fitness values. The *roulette wheel* selection is among most popular, but better performances can be achieved using the *tournament selection*. Tournament selection for selecting one parent looks like follows:

Choose  $k$  individuals from the population at random;  
 Select the best individual from the pool/tournament  $x$ ;  
 Parent  $\leftarrow x$ ;

The selection of  $k$  individuals can be performed either with or without replacement [6]. In the first case, once selected individuals are candidates for other tournaments as well. Although the run duration is approximately the same in both cases, tournament selection without replacement requires less population [6].

### 3.5. Elitism

It is possible to replace the whole existing population with a new generation, but it may result in losing the best possible solution. By keeping the best individual(s) from previous generation, convergence speed may be drastically improved. Elitism demands caution. Too many elite individuals can actually slow down convergence because more generations are necessary to investigate new individuals. Number of individuals kept from previous generation shouldn't be greater than 10% of the population size.

### 3.6. Genetic Operators

Two most elementary genetic operators are *crossover* and *mutations*. There are many ways to implement crossover. Among most popular are *one-point crossover*, *multipoint crossover*, *order crossover*, *cycle crossover* etc [3].

One-point crossover for matrix representation is described in Figure 4. Child1 is generated by coping first two rows from Parent1. Next, items from Parent2, which are not included in the first two rows of Parent1, are copied sequentially to the last two rows of Child1. By swapping the role of Parent1 and Parent2, Child2 may be generated in the same way.

Parent1				Parent2			
1	15	3	5	14	4	1	2
11	12	4	9	10	9	7	15
10	14	2	6	3	5	12	16
13	8	16	7	6	8	11	13

Child1				Child2			
1	15	3	5	14	4	1	2
11	12	4	9	10	9	7	15
14	2	10	7	3	5	11	12
16	6	8	13	6	13	8	16

Figure 4. One-point crossover for matrix representation

Described method is used to mate two individuals from the old population, producing two children. Individuals are mated group by group and serial numbers of slots are used to identify slots in unique way. New generation can include both of generated children or the one with better value of the fitness function. Crossover is performed on selected pair of individuals with probability  $p_{crossover}$  (*crossover*

*probability*). Generated individuals may be transformed by mutation. Just like crossover, it is possible to define mutations in many different ways. Swapping two randomly selected slots in a randomly selected group is one way of performing mutation on an individual. The other way could be exchanging whole rows or columns. Whether a new generated individual will be transformed by mutation or not, is defined by *mutation probability*. Mutation probability should be big enough to bring in new quality in the population, but too big mutation can turn GA in *random search*. Mutation can be *blind* and *intelligent* [5]. The difference is in fact that intelligent mutations are performing only changes which do not decrease fitness value of an individual. Mutations can also be used as a "repair function" [2]. The main idea is to use some sort of transformation during fitness evaluation which can try to eliminate conflict as soon as it is detected. This combination can drastically improve software performances. One more improvement can be introduced. Crossover and mutation probability are not necessary constants in time. As the number of generations grows, the crossover probability is progressively decreased, while the mutation probability is increased [7]. Variable probabilities can help to speed up slower part of GA (N-P on Figure 5).

## 4. EVALUATION

Created software was tested on real life data. We created timetable for Computer Science Department at "Faculty of Electronic Engineering", University of Nis. In Table 1, some basic characteristics of that system are presented.

Num of Teachers	40
Num. of Subjects	43
Classes (num. of student groups)	10
Total num. of lessons	194

Table 1. Basic characteristics of used input data

Optimal parameters for GA depend on specific problem. We achieved the best performances using values shown in Table 2.

Parameter	Value
Population size	25
Mutation probability	0.2 (increasing to 0.5)
Tournament size $k$	2
Crossover probability	0.8 (decreasing to 0.5)
Number of elite individuals	2

Table 2. Optimal parameters for GA

Combining function evaluation with mutations we were able to eliminate most hard conflict as soon as we detected those. We also didn't choose initial population completely randomly. Some of preferred characteristics have been incorporated in all of initial individuals assuring that their offspring has those characteristics. By doing that, we were able to drastically decrease number of generations necessary to achieve timetable without hard constraints, thus we shortened time of program execution.

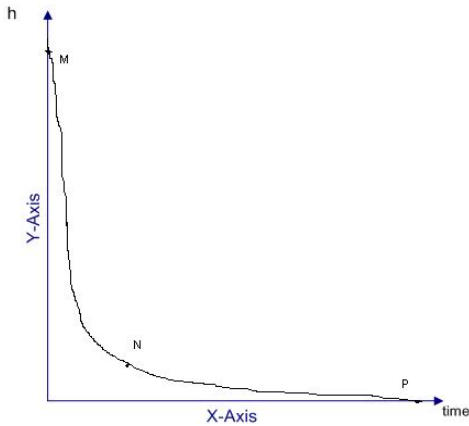


Figure 5. Convergence speed of genetic algorithm

## 5. CONCLUSION

Genetic algorithm and its usage in solving timetabling problem are presented in this paper, as well as several optimization methods. Our experiments showed that GA generally can quickly find "good" solutions which are *close to correct* (M-N on Figure 5). Different kinds of optimizations are necessary to improve convergence speed in slower part of the algorithm (N-P on Figure 5). Our future work will be focused on improvement of that part of the algorithm.

## REFERENCES

- [1] Marte M., "Models and Algorithms for School Timetabling –A Constraint-Programming Approach", Ph.D. Thesis, Ludwig-Maximilians-Universit at Munchen, 2002.
- [2] Fernandes C., Caldeira J.P., Melicio F. and Rosa A., "Infected Genes Evolutionary Algorithm for School Timetabling", WSEAS press, pp 245-249, 2002.
- [3] Petres Z., Gy"ori S. and V"arkonyi-K"oczy A.R., "Genetics Algorithms in Timetabling. A New Approach", MFT Periodika, 7, 2001.
- [4] Goldberg D.L., . *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
- [5] Di Stefano C. and Tettamanzi A.G.B., "An Evolutionary Algorithm for solving the School Time-Tabling Problem", Evo:Workshops 2001, pp. 452-462, 2001.
- [6] Sastry K. and Goldberg D.E., "Modeling Tournament Selection with Replacement Using Apparent Added Noise", IlliGAL Report No. 2001014, 2001.
- [7] Fang H-L., "Investigating genetic algorithms for scheduling", MSc Thesis, University of Edinburgh, 1992.