Solving Timetable Scheduling Problem by Using Genetic Algorithms

Branimir Sigl, Marin Golub, Vedran Mornar
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb, Croatia
branimir.sigl@bj.hinet.hr, marin.golub@fer.hr, vedran.mornar@fer.hr

Abstract. *In this paper a genetic algorithm for solving timetable scheduling problem is described. The algorithm was tested on small and large instances of the problem. Algorithm performance was significantly enhanced with modification of basic genetic operators, which restrain the creation of new conflicts in the individual.*

Keywords: timetable scheduling problem, genetic algorithm, 3D representation

1. Introduction

This article describes an implementation of genetic algorithm on timetable scheduling problem. The timetable scheduling problem is common to all educational institutions. Main algorithm goal is to minimize the number of conflicts in the timetable. Reduction to encoding of search space was also implemented. The algorithm was tested on small and large timetable problems at Faculty of Electrical Engineering and Computing (FER) in Zagreb. The program interface was developed in C#. The core of genetic algorithm was developed in C++ with STL (Standard Template Library) support.

2. Timetable Scheduling Problem

The scheduling problems are essentially the problems that deal with effective distribution of resources. During the scheduling process many constraints have to be considered. Resources are usually limited and no two tasks should occupy one particular resource at the same time. For most of the scheduling problems it has been shown that they are NP-hard, and that they can not be solved in polynomial time using a deterministic algorithm.

School timetable scheduling problem presents a set of tasks (classes) and a set of resources (rooms, groups, instructors). Every task requests some resources for its realization and has the exact length. The set of timeslots when a class can be scheduled is also determined. The goal is to assign those tasks to their resources while satisfying all of the hard constraints – no resource should be allocated by multiple tasks at the same time.

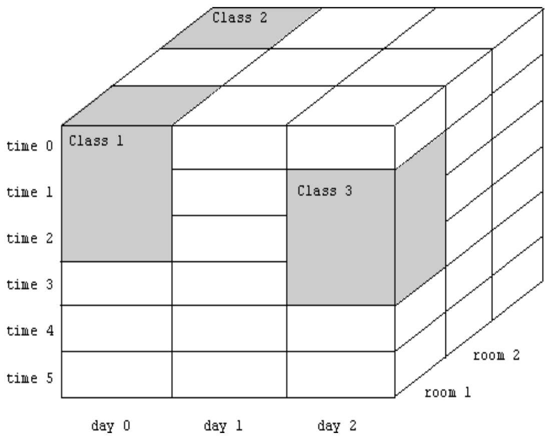


Figure 1. Timetable presented with 3D structure

Scheduling a timetable could also be represented like special class of 3D cutting problems. The timetable could be presented as a 3D structure. The dimensions of 3D timetables are: days (x-axis), timeslots (y-axis) and rooms (z-axis). The classes are shown as cubes, which should be placed in a 3D timetable structure (Fig. 1) [7]. The scheduling is a process of placing those cubes into a timetable, in the way that no conflicting classes (which allocate the same resource, a student group or an instructor) are placed in the same timeslot.

The timetable scheduling process could be formally defined with binary variables x_{cdtrgi}, which have the value of 1 if and only if instructor i lectures the class c on day d at time t, for group g in room r.

The timetable should satisfy the following conditions:

a) Group *g* can attend only one class at one time.
b) Instructor *i* can teach only one class at one time.
c) In room *r* only one class can be taught at one time.
d) All lectures should be kept exactly once.

A GUI (Graphical User Interface) has been developed to facilitate the input of data. For each class the following can be set:

- days and times when the class *could* be placed;
- rooms where the class *could* be placed;
- number of rooms occupied by a class simultaneously
- groups of students that attend the class;
- instructors that teach the class.

The theoretical problem size can now be reduced. As the groups and instructors are intrinsic part of a class definition, indices *i* and *g* are eliminated. The information about groups and instructors is not discarded, however. It is stored with the variable to be utilized later when generating conflicts.

Furthermore, since many combinations of indices *c,d,t,r* are impossible, only the possible combinations are generated.

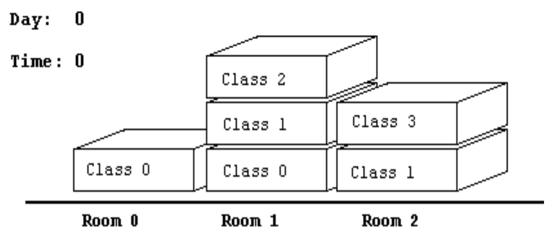
Since one class can utilize more than one classroom (e.g. a larger group occupies two PC labs at he same time), the index *r* of variable x_{cdtrgi} actually denotes one of the possible combinations of the room allocations.

The pseudocode for the variable generation is:

```
for each class c {
    generate all possible room combinations
    for each possible (day,time) pair {
        for each r in room combinations {
            create variable $x_{cdtr}$
        }
    }
}
```

To facilitate the generation of conflict, three auxiliary 3D structures are created. Each structure represents a special type of view on the timetable: from the aspect of room, group and instructor. From every view new constraints can be identified. *X* and *y* axes of all three auxiliary structures represent the day and time. The *z*-axis is different in every structure, representing rooms, groups and instructors, respectively. During the constraint generation process, each variable is positioned, for all possible day-time pairs, at the appropriate *z* coordinate, which denote rooms, groups or instructors allocated by the corresponding class. After filling of all the data, each *x*-*y*-*z* coordinate is checked[1]. If more classes compete for a particular resource, a new constraint has to be generated. This process is analogue to reducing of resource to only one task in a single timeslot. The solver is now assured that only one variable will take (for example) a particular room in a single day-time combination. All of those bound could be represented as:

$$\sum x_{class} \leq 1 \qquad (1)$$



```
Generated conflicts:

Room1:     Class0 + Class1 + Class2  <=  1
Room2:     Class1 + Class3           <=  1
```

Figure 2. Generation of conflicts and bounds

3. The Genetic Algorithm Implementation

Genetic algorithms are adaptive systems inspired by natural evolution. They can be used as techniques for solving complex problems and for searching of large problem spaces. Genetic algorithms are belonging to guided random search techniques, which try to find the global optimum. J.H. Holland presented this concept in early seventies. The power of genetic algorithms and other similar techniques (simulated annealing, evolutionary strategies) lies in the fact that they are capable to find global optimum in multi-modal spaces (spaces with many local optimums). Classical gradient methods will always gravitate from starting position to some local optimum, which could also be global, but it can not be determined for certain. Genetic algorithms are working with the set of potential solutions, which is called *population*. Each solution item (*individual*) is measured by *fitness function*. The fitness value represents the quality measure of an individual, so the algorithm can select individuals with better genetic material for producing new individuals and further generations.

The simulation of evolution allows survival of better individuals and extinction of inferior ones. Evolution’s goal is to find better individuals in each generation. The process of evolution is maintained by selection, crossover and mutation. In terms of genetic algorithms those processes are called *genetic operators*. The selection chooses superior individuals in every generation and assures that inferior individuals extinct. The crossover operator chooses two individuals from current population (*parents*) and creates a new individual (*child*) based on parents’ genetic material. Selection and crossover operators will expand good features of superior individuals through the whole population. They will also direct the search process towards a local optimum. The mutation operator changes the value of some genes in an individual and helps to search other parts of problem space.

In the algorithm presented here, each individual in the population represents one timetable. The algorithm starts from an **infeasible** timetable, and tries to get the feasible one.

In a timetable, every class can be placed only once in the 3D timetable structure. This could be ensured with generation of a new constraint for each class that should be scheduled. These extra constraints would just enlarge the problem size and the number of constraints that should be checked. Because every class can have only one variable set to 1, individuals can be generated in such way that every gene in an individual represents one class. The value of a gene will be the ordinal of a binary variable belonging to that class.

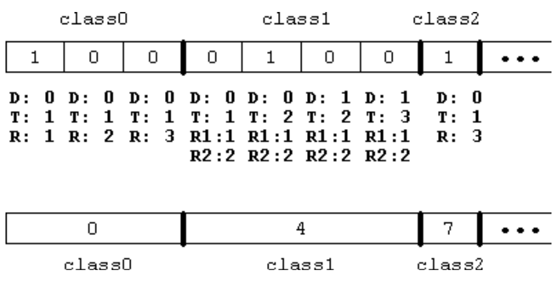


Figure 3. Encoding of Individuals

The fitness value of an individual is calculated as

```
fitness(individual) =
(Number of conflicts)*K + Quality     (2)
```

The number of conflicts shows how many constraints have been violated within the current individual. When an individual reaches zero conflicts, that means that it represents a feasible timetable and that there are no collisions of classes [1].

The quality of the timetable is determined by earliness of scheduled classes. Students have better ability to learn in morning hours and after that, the interest for learning is continually decreasing. That is why the best quality value is set to the early hours and worse values are set for late hours. The genetic algorithm will try to schedule classes as early in the morning as it can, indirectly minimizing the number of holes in a student's schedule.

The main goal of the genetic algorithm presented here is to achieve a feasible timetable. That is why the feasibility function will mainly try to minimize the number of conflicts in an individual. This is achieved by multiplying the number of conflict by a large constant K. The quality of the timetable is of the lesser importance. Individuals in a population are sorted by ascending value, so the best individual has the smallest fitness value.

The program uses eliminating selection, which chooses and eliminates bad individuals from the current population, making room for new children that will be born from the remaining individuals. The probability of elimination increases proportionally with the fitness value of the individual. As the remaining individuals are better than the average of the population, it is expected that their children will be better as well.

There is some probability (though very small) that eliminating selection deletes the best individual. That would ruin the algorithm efforts and put its work back for some number of generations. Therefore, protection mechanism for best individuals has to be made, so the good genetic material is sustained in population. It is called *the elitism*. The authors’ choice was to keep just the top one individual.

The reproduction operators constitute a very important part of genetic algorithms. Those operators make use of good individuals (which remained in population after selection) and construct new, better individuals and overall population.

The crossover operator operates on individuals (called *parents*) and make new, *child* individual from their genetic material. This operator fills up empty places in population that remained after elimination. If parents are good, it is likely that their child will also be good. ***Uniform*** crossover operator was chosen as the best option for this kind of problem [8].

```
for each gene in (parent$_1$ , parent$_2$){
   if(parent$_1$[gene]==parent$_2$[gene]){
      child[gene]=parent$_1$[gene];
   }else{
      child[gene] =
      random(parent$_1$ ,parent$_2$)[gene];
   }
}
```

Uniform crossover operator checks all genes of both parents. If parents have equal values of a gene, this value is written to the child. If values from parent genes differ, then the algorithm randomly chooses one parent as a dominant one and takes its gene.

The program uses simple roulette wheel parent selection algorithm. The probability of selection of one individual is proportional to it's fitness value [4].

Cumulative fitness values are used for each individual by the formula:

$$q_k = \sum_{i=1}^{k} fitness(individual_i), \quad D = \max(q_k) \quad (3)$$

where k=1, 2, …. *POPULATION_SIZE*.

The algorithm generates a random number r from the interval *(0, D)* and selects an individual which satisfies the condition:

$$\max q_k <= r. \quad (4)$$

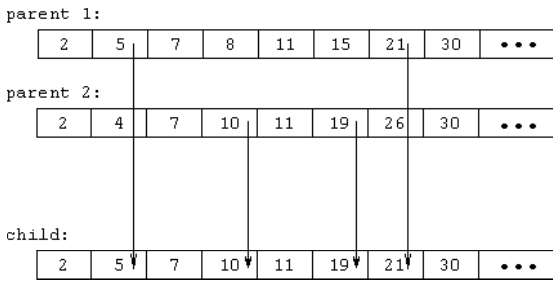


Figure 4. Uniform crossover of individuals

The mutation is also a typical operator for the genetic algorithm. It takes one or more genes from an individual and changes its value. The probability of the mutation is an input parameter for genetic algorithm. The presented algorithm iterates through every gene of every individual in the population. For each gene a random number form the interval (0, 1) is generated. If the generated value is smaller than the given probability of the mutation (p_m), the gene changes value to a random value which denotes a different day-time value or room combination.

```
for each gene in individual{
   if(p$_{(Random)}$ < p$_m$){
      gene = get random value from
                possible values list;
   }
}
```

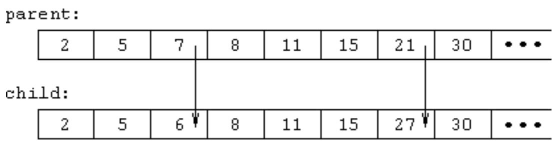


Figure 5. Mutation of individuals

4. Improving genetic algorithm behavior

The main idea for improving operators was to prohibit the introduction of new conflicts. Basic operators did not take into account whether they make individuals with more or with less conflicts [5]. The outcome of the algorithm utilizing such basic operators was noticeably poor. The decision was made to add some programming logic to the operators and to use different selection algorithm.

The first step of the improved crossover operator simply copies equal genes from parents to the child individual. No additional conflicts can be added through copying those equal values. Different parent's genes are specially marked and delegated for further processing. In the second step, the crossover operator checks the list of equations for each marked gene in child individual and counts the number of potential conflicts generated for both parent's choices. The gene from a parent that generates fewer conflicts is chosen, so no additional conflicts are generated in addition to the conflicts caused by the parents.

Such a modification was not introduced to the mutation operator. Similar functionality of the mutation operator would probably lead the whole population to the local optimum. In most cases the mutation operator generates inferior individuals, but most of its genes are good. Mutated genes help us to explore other parts of searching space and to avoid reaching of local optimum value.

Additional conflicts in population will be generated when two equal individuals appear in the mating process as parents. In that case one of the parents will be mutated and the child will be randomly generated. In this procedure a bad child individual will be created, but it will not spoil the population. Quite the reverse, it will bring new genetic material to the saturated population.

The improved algorithm uses tournament eliminating selection [5]. This kind of selection ensures elitism. Consequently, the top individual can not be changed. The tournament selection allows greater population size without slowing down the algorithm. Better results were obtained with population of much greater size, in contrast to the basic algorithm, where enlargement of the population resulted in significant performance degradation.

5. Experimental Results

For experimental purposes data from Faculty of Electrical Engineering and Computing (FER) in Zagreb was used [7]. Algorithm was tested on the small and large instances of problem (Table 1). The large problem is a full size FER schedule for the autumn semester. The small size problem was obtained from the large size with exclusion of about 70% of classes from the scheduling process. The small problem was solved without conflicts. When solving the large size problem, the basic algorithm stopped at about 95 conflicts. With intelligent operators, algorithm reached about 20 conflicts.

Table 1. Size of the problem

	SMALL	LARGE
Classes / Size of individual	227	770
Rooms	27	41
Groups	55	114
Instructors	46	157
Number of binary variables	16103	35026
Number of bounds	4345	10898
Population size (classical)	256	256
Population size (improved)	5120	5120

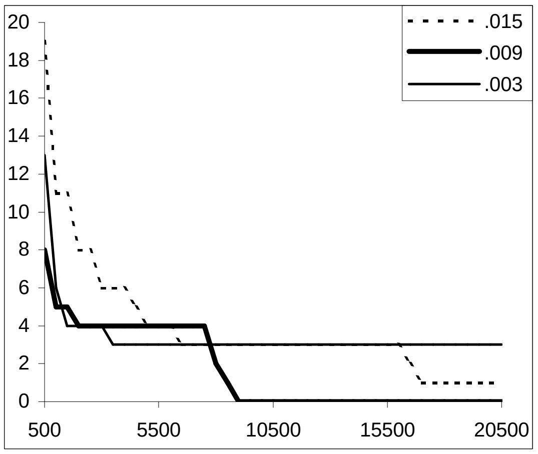


Figure 6. Evolution process for the small size problem depending on mutation probability

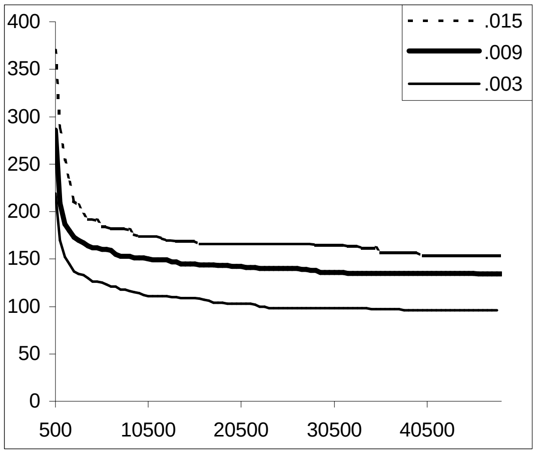


Figure 7. Evolution process for the large size problem depending on mutation probability

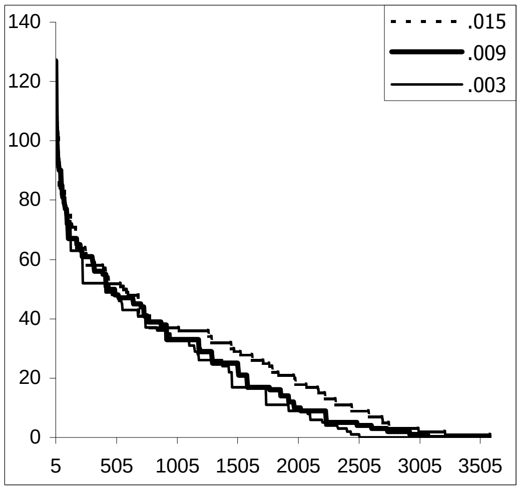


Figure 8. Evolution process with improved operators for the small size problem

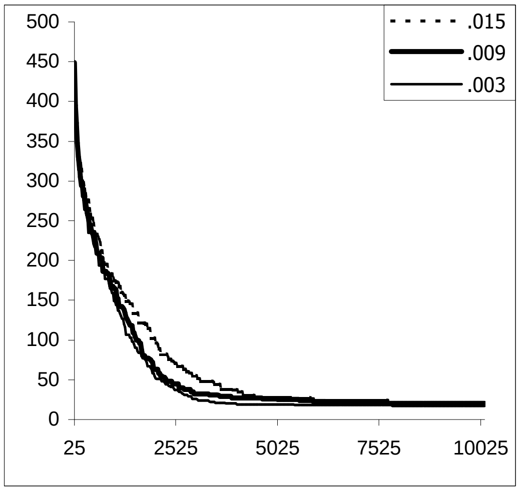


Figure 9. Evolution process with improved operators for the large size problem

Figures 6. and 7. show the results of genetic algorithm with basic operators. This algorithm experiences problems with convergence and saturation of the population. After a certain number of conflicts had been reached, very small improvements were achieved through algorithm running time.

The algorithm with improved operators (Figure 8, and 9.) shows much better results. Faster convergence and fewer conflicts were achieved in a lesser amount of running time.

8. Conclusion

The initial scheduling problem with large number of binary variables has been reduced to the acceptable size by eliminating certain dimensions of the problem and encorporating those dimensions into constraints. The grouping of several binary variables into one gene value significantly reduced the individual size. Now it is possible to try to solve the full size problem (problem of the whole FER schedule) with genetic algorithm approach. Such a representation of the scheduling problem achieves the acceptable algorithm speed, so small size problems are solved in tens of seconds. Significant improvements have been achieved by using intelligent operators. The intelligent algorithm converges much faster then the basic algorithm and represents a good starting point for complete solving of the full scale problem.

To completely solve the full scale problem, further algorithms improvements will have to be made. When generating the constraints, it could be useful to sign each one, so no constraint will be set (and checked) twice. Individuals should be generated in such way that classes which are more difficult to schedule occupy the front genes of an individual, while classes easier to schedule should occupy the back genes. This would be useful for intelligent crossover operator, which sets and checks the conflicts from front to back of the individual. Also, a parallel computing approach should be tried, so checking space of the problem could be widened. Each thread could start with different initial population and the quality of solution is expected to be better.

9. References

[1] D. Abramson, J. Abela, “A Parallel Genetic Algorithm for Solving the School Timetable Problem”, 15. Australian Computer Science Conference, Hobart, Feb 1992.

[2] L. Davis, "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York, 1991.

[3] D.E. Goldberg, "Genetic Algorithms in Search", Optimization and Machine Learning, Addison-Wesley, 1989.

[4] M. Golub, D. Jakobović, “Genetic algorithm", parts I and II (in Croatian), Faculty of electrical engineering and computing, Zagreb, 1997/2002, available via ftp from: http://ww.zemris.fer.hr/~golub/ga/ga.html

[5] M. Golub, S.Kasapović, "Scheduling Multiprocessor Tasks with Genetic Algorithms", Proceedings of the IASTED International Conference Applied Informatics, February 18-21, 2002., Insbruck, Austria, pp. 273-278, available via ftp from: http://www.zemris.fer.hr/~golub/clanci/ai2002/ai2002.doc

[6] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolutionary Programs", Springer-Verlag, Berlin, 1992.

[7] V. Mornar, “Algorithms for some classes of cutting stock problems", PhD dissertation (in Croatian), Electrotechnical faculty, University of Zagreb, Zagreb, 1990.

[8] R. Weare, E. Burke, D. Elliman, “A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems”, available via ftp from: http://www.citeseer.com