Genetic Algorithms in Timetabling. A New Approach.

Sándor Győri
gyori@szit.bme.hu

Zoltán Petres
petres@szit.bme.hu

Annamária R. Várkonyi-Kóczy
koczy@mit.bme.hu

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Műegyetem rkp. 9., Budapest, Hungary, H-1521

Abstract — The timetabling problem comes up every year in educational institutions, which has been solved by leveraging human resource for a long time. The problem is a special version of the optimization problems, it is computationally NP-hard. Although, there are some attempts to apply computer based methods, their use is limited by the problem's complexity, therefore Genetic Algorithms were applied, because they are robust enough in such a huge problem space. In this paper a new and more flexible timetable representation, the set representation is introduced which meets the demands better than former ones. The proposed method proved to be efficient in real life application of a secondary school, as well.

I. INTRODUCTION

The timetabling problem, which has an important role typically in education, is a special version of the optimization problems found in real life situations.

The timetabling problem has always been solved by leveraging human resource in educational institutions. During the process, numerous aspects have to be taken into consideration. Almost a week of work of an experienced person is needed to produce a timetable for an average institution and the result is often not satisfactory; it does not meet all the requirements. What is more, when the preconditions change, the whole work becomes unusable, and has to be restarted from scratch. The problem—as almost all optimization problem—is computationally NP-hard. Therefore, only the important conditions can be considered during the manual arrangement process, but it is still extremely complex to find the optimal solution of this reduced problem.

Thus, a good timetable generator software that would take into consideration not only the essential conditions necessary for a usable timetable, but also other important didactic and organizational requirements would be very useful. This became reality by the tremendous growth of computing capacity. In case of optimization problems [1, 2]—as the problem described in this paper—genetic algorithms (GA) [3, 4] proved to be sufficient.

The paper is organized as follows: In Section II. the timetabling problem including hard and soft constraints and classical representations is discussed. Then the set representation is introduced in Section III. Genetic Algorithms are presented as a solution for the problem in Section IV. and various operators and improvement technics are analyzed in Section V. Section VI. is devoted to experimental results.

II. THE TIMETABLING PROBLEM

The timetabling problem comes up every year in educational institutions. Students, teachers, lessons and classrooms have to be arranged optimally. It is very difficult to define how good a potential timetable is, but much easier is to declare when a timetable is unusable as it is always exact. By consulting to a person experienced in timetabling problem, the situations that should be avoided in order to get a nearly optimal result can easily be specified, namely the constraints should be satisfied by the timetables. Hard and soft constraints should be distinguished.

A. Hard constraints

Hard constraints have to be taken into consideration very strictly, because the timetables that violate just one of these are unusable. The finite "resources" belong to this group.

The class clash situation is when a student of a class should participate in more than one lesson at the same time. Of course, the splitted lessons do not cause class clashes.

The teacher clash is similar to the class clash, but in this case a teacher should give more than one lesson at the same time. The lessons held for merged classes by the same teacher do not cause teacher clashes.

Finite room capacity: It is not possible to give two or more different lessons for two or more different classes at the same room. It is expedient to divide the different rooms into groups according to their similar functionality and to use these groups during the optimization process.

Teacher's (strict) availability: A teacher is not necessarily available in the whole educational period, so it is important to know when a teacher is available and arrange the lessons according to this.

B. Soft constraints

The didactic and organizational constraints represent the group of soft constraints. The timetable that violates these constraints are still usable, the lessons can be held according to it, but it is not convenient for either students or teachers, and it also makes more difficult to understand the lessons.

Lessons held at early morning and long breaks between two lessons are not acceptable at all in primary schools, and hardly tolerable in secondary schools. In most cases these problems can be avoided.

Multiple lessons: Because of didactic purposes, in case of some subjects it can be useful to hold two, three, or maybe more lessons consecutively. These subjects could be art, physical education or in specialized classes mathematics, history, etc. The fundamental unit of time in timetable representations is one lesson, so it is always necessary to check that the members of multiple lessons should be placed next to each other.

The situation when same type of lessons spontaneously come next to each other is called bunchyness in a day. It is important to distinguish bunchyness from the multiple

lessons' situation as in the first case it is not accepted while in the second case that is what we want.

Same type of lessons' equal distribution in a week: Because of pedagogical reasons it is expedient to distribute the lessons of the same kind equally in a week.

Equal number of lessons per day: The difference between the maximum and the minimum value of the number of lessons per day shows the equality. If the number of lessons for a week is not dividable by the number of days ± 1 lessons fluctuation is allowed.

Teacher's soft availability: It is possible that a teacher ask not to have lessons in some timeslots. This demand should be satisfied just in case when the timetable does not violate any other kind of hard or soft constraints including teacher's strict availability, too.

C. Classical representations

There are several ways how to represent a timetable. From the aspect of a human observer the most important is to get the answer as easily and as fast as possible for the occupations of classes, teachers or, maybe, rooms at a given time. Another important question is that how efficient the representation suitable for human usage is for computer processing.

The traditional, two dimensional timetable representation, usually used during the manual timetable arrangement process, has the different classes on the horizontal axis, while the time periods for the lessons on the vertical axis. In this matrix the item (i, j) contains the teachers who give a lesson for class i at time period j.

There is another two dimensional representation which has the time period on the horizontal axis and the teachers on the vertical axis. The item (i, j) of the matrix contains those classes which has lesson at time period j held by teacher i.

The first representation is efficient in the class based searches, while the second one in the teacher based searches, so in practice, both representation is done.

Both representation have the advantage that they ensure an implicit constraint: in the first case the class clash, while in the second the teacher clash is obvious immediately.

Unfortunately, these two representations do not support the splitted lessons, because they do not distinguish the concept of teacher and subject at all. Thus, the situation, when a class has a lesson held by two foreign language teachers, is equivalent with another situation, when a class has a lesson held by a mathematics and an art teacher, but while the first case can be possible, the second definitely not.

The distinction between the concept of teacher and subject is usually solved by virtual teachers proposed in [5]. More virtual teachers can belong to a real one. A virtual teacher is allowed to teach only one subject, so the problem is reduced to the main concept. The real teacher can give just maximum one lesson at one time in this case, too. With a real teacher–virtual teacher association, this can be checked easily.

The distinction between teacher and subject is also indispensable in the following case: a teacher gives several lessons in different subjects to the same class and the equal distribution of lessons are checked.

Unfortunately, the solutions proposed above do not satisfy all the real life situations, as they do not allow the case when more teachers give lessons to several classes at the same time. This situation is very common e.g. in the foreign language education: five teachers give different kind of language lessons to two classes (e.g. German beginner, German advanced, Russian, Italian, Spanish). The following cases have to be ensured: both classes have to have the same foreign language classes at the same times, and the five teachers have to be available for education at the times when the classes have the foreign language lessons.

D. Methods

Because of the problem's complexity, the classical algorithms are less or not efficient at all, despite that there are some initiations based on known algorithms as heuristic search [6] or modified back-track. The Slovakian aSc group [7] offers a usable timetable arrangement software which uses direct search algorithm. During the timetable arrangement process, if numerous requirements are taken into consideration the problem's complexity can easily grow out of the manageable domain, and then direct search algorithm does not provide satisfactory results within reasonable time. Simulated annealing (SA) algorithm is used for timetabling problem by A. Abramson in [8]. Tabu search (TS) is introduced for a possible solution for timetabling problem by Hertz [9] and de Werra [10]. Unfortunately, they are not effective enough in such a big problem space and they work on just one timetable. Simulated annealing and tabu search are examined for timetable problem by Colorni, Dorigo and Maniezzo in [11]. The efficiency and the convergence speed of these algorithms are compared to the similar values of genetic algorithm (GA) and the result was that the values of genetic algorithm and a specialized version of tabu search are almost the same while the ones of simulated annealing much worse than the other two.

III. "SET" REPRESENTATION FOR TIMETABLING

In order to improve the above mentioned imperfections of the known representation methods in this section a new representation, the "set" representation is introduced. The set representation tries to solve the above mentioned problems and it allows the biggest freedom in class merging and splitting and in teacher–subject association.

The smallest data unit is the set. This structure can consist of any number of classes, teachers and rooms. These sets are indivisible during the optimization process, they are moved together in the different timeslots. Its meaning is the following: the teachers given in the set hold lessons to the classes in the given rooms in one same timeslot. The main concept does not define which teacher gives lessons to which classes

Day 1			Day 2			⋯
Lesson 1	Lesson 2	⋯	⋯	⋯	⋯	⋯
Set_1: $\text{class}_{c_{1,1}}, \text{class}_{c_{1,2}}, \ldots$ $\text{teacher}_{t_{1,1}}, \text{teacher}_{t_{1,2}}, \ldots$ $\text{room}_{r_{1,1}}, \text{room}_{r_{1,2}}, \ldots$	Set_{N+1}					
Set_2	Set_{N+2}					
⋮	⋮	⋱				
Set_N	Set_{2N}					

Fig. 1: Set representation

and where the lessons are held. This method allows to solve such problematic situations as class merging and splitting in a very flexible way. For example, if two classes have PE lesson in the same time and they are divided into two groups according to their sex, then we simply add the two classes, the two teachers and the necessary rooms to a set, thus defining the required constraint.

This timetable representation has only one dimension, the time. The sets have to be placed in the different timeslots—which indicate the possible time periods for classes—in the time line (see Fig. 1). With $c_{i,j}, t_{i,j}, r_{i,j}$ values the serial numbers of classes, teachers and room groups that consist of set i are indicated. The values are chosen from the corresponding serial number sets. All the timeslots have fixed (N) number of sets, which is very important in the adaptation of the genetic algorithm (see later). A set can be empty, as well (it is similar to the case when a cell of the matrix in the classical, two dimensional representation is empty). Thus, the number of sets that have real information is not constant any more. In this representation the class and the teacher clash is not as obvious as in the classical representations so in every timeslot the sets should be checked whether they consist of the same classes or teachers.

This representation is very convenient for computer processing, but for a human reader it is hard to understand. So, it is expedient (typically at the end of the optimization process) to map the sets placed in different timeslots to one of the classical representations. In order to make a one-to-one mapping, it has to be exactly known that which teacher gives lessons in what subject to which class, thus the sets have to be completed with additional information as the subjects and also as the associations between the classes and teacher, subject, room group triplets.

IV. THE GENETIC ALGORITHM

Genetic algorithms (GA), which simulate the inheritance of living beings, is a widely used method for solving optimization problems. [4]

GA performs a multi-point search in the problem space. On one hand it ensures the robustness as if one searching track sticks in a local minimum it does not mean that the whole algorithm fails, while on the other it may give not just one, but more nearly optimal solutions for the problem from which the user can select. The algorithm during the optimization process uses deterministic and stochastic methods, and thank to this, it can also solve those problems, such as the timetabling problem, which seems to be too hard, too complex for any other kind of optimization methods. Its two main operations are the crossover and the mutation.

In case of the timetabling problem the consistence of the individuals' genes is extremely important. The consistence means that each type of information have to be presented just once in the genotype, so, instead of using destructive operators, just the mixing operators are usable. In the nature, this is realized in such a way that the information takes place in predefined locations in the chromosomes and during the crossover just these chromosome parts can change. The system is redundant enough to tolerate if some parts are duplicated or missing and the mapping between the genes and the properties is not mutual, as well. Despite this the consistence is corrupted sometimes in the nature, too, but it can not be allowed in our case. This could be best illustrated by considering, e.g. that the number of history lessons cannot be increased at the expense of physics lessons.

A. Linearization

Our genetic algorithm does not care for the internal structure of the individuals and for the meaning of data on which it executes the operators. Moreover, it is expressly disadvantageous if an individual—because of its complex internal structure—cannot be divided at any wanted point and then paste together with another piece. To avoid these problems the timetable has to be coded and on this coded form is the algorithm used.

In the set representation of timetabling the order of the sets in a timeslot is optional, the only important thing is that in which timeslot a set can be found. The conversions done on the introduced representation which make a linear structure from a two dimensional one by keeping the time dimension is suitable for our problem.

Vertical linearization: A simple, linear structure is got by taking the sets from the time axis beginning with the first set of the first timeslot, then the second set of the first timeslot, and so on until the last set of the last timeslot. For the one-to-one mapping it is necessary that each timeslot has constant number of sets. Using "empty sets" is a possible solution for this problem. If the class merging is not allowed and all the classes have lessons in all the timeslots, the number of sets in each timeslot is equal with the number of classes. If a class has not got a lesson in a timeslot an empty set can be added to maintain the needed number of sets. With empty sets the class merging situation can be solved, too. Of course, if several samples of a set are used the same number of empty sets have to be added to maintain the structure.

Horizontal linearization: There is another possible way to get the linear structure of the sets: firstly the first sets of each timeslot is taken, then the second sets of each timeslot, and so on (see Fig. 2).

The vertical linearization emphasizes the relatedness of the sets of the same timeslot, as during a crossover the composition of the first or the second part of the timeslots of an individual remains (with great probability it transmits the clashes to the next generations). While the horizontal linearization emphasizes the connection of the individual sets to a timeslot, as during a crossover the places of the first or the second part of the sets of an individual remain (with great probability it transmits the teacher's strict availabilities

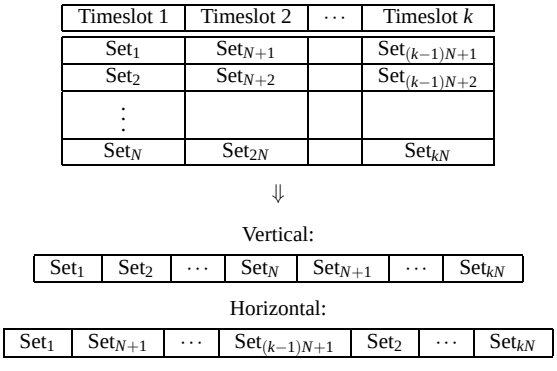

Fig. 2: Vertical and horizontal linearization

to the next generations). Despite of the important difference the efficiency of the algorithm does not significantly depend on the chosen linearization method.

B. The Fitness function and selection

The viability of the individuals can be calculated from its penalty values. The mapping is done by the fitness function (FF). The efficiency of the genetic algorithm mainly depends on the used coding method (in this case it is the set representation for timetabling) and on the fitness function [12, 13].

The best results can be obtained by the usage of reciprocal functions which is confirmed by our experiences and by several papers (e.g. [14]). Our examinations were made with the following functions: $\frac{1}{1+x}$, $\frac{1}{1+x^2}$, $\frac{1}{1+\sqrt{x}}$, $\frac{1}{1+\ln x}$. According to the test results—corresponding with the works of [14] and [12]—the fitness function $\frac{1}{1+x^2}$ ensures the best convergence. An explanation for it can be that at the beginning when the penalty values are high the differences between the fitness values are not sufficiently significant to make distinction between the competing individuals, while getting closer to the optimum the penalty values become much lower, the fitness function discriminates more sharply, thus it emphasizes the more viable individuals. In this way the chromosomes containing excellent parts also get some chance to transmit their genetic substrate in spite of the fact that other parts are worse than the average.

The individuals have to be selected from the previous generation according to their weighted probability of viability. The roulette wheel selection is a possible method for this. With the tournament selection which is an improved version of the roulette wheel selection the convergence speed can be increased by approximately four times.

V. GENETIC OPERATORS

Crossover is one of the most essential genetic operators. Its task is the realization of the deterministic search, it tries to advance in the problem space by applying the extant knowledge. From two initial individuals it generates a new one which genetic substrate is the combination of the genes from the initial individuals. In practice from the two initial individuals four new individuals are generated, and, on the whole, the genetic substrate of the four children covers the genetic substrate of the parents. Thus, the probability to get a better individual is favorable.

The simplest, one-point, consistence preserver crossover operator is the following: call the initial individuals A_1, A_2 and the four children generated from them B_1, B_2, B_3, B_4. Select a crossover point randomly (the same in both initial individuals) and then copy the first part of A_1 to B_1. Next, choose those items from A_2 which are not present in the first part of A_1 and fill the end of B_1 with them. To generate B_2, copy the second part of A_1 to the end of B_2, then choose those items from A_2 which are missing at the end of A_1 and put them one after the other at the beginning of B_2. By swapping the role of A_1 and A_2 the children B_3 and B_4 can be generated (see Fig. 3). In order to preserve consistence the internal structure of A_2 can be hardly transmitted into B_1 and B_2. Namely, when those items are chosen from A_2 which are not present in the first part of A_1, a "spongy", ragged structure is got that has to be compressed by neglecting the empty places in order to put it at the end of B_1. During this procedure, such items may get next to each other that were not in neighborhood originally. The two-point crossover is a similar operator, at the beginning two crossover points are selected, then the same procedure is used as in the case of the one-point crossover.

Much more complicated methods are used in the operators worked out for permutations which are developed by David E. Goldberg et al. [3]. Important that these operators can be used just for permutations, i.e. for those genetic substrates whose items are individuals and free of same items. To use these operators in the set representation of timetabling subserial numbers have to be introduced, so in this new numbering the same type of sets has individual serial numbers.

For the order crossover (OX) two crossover points are selected randomly. By using the previously introduced notation, the genetic substrate of A_2 is copied to B_1, then the items found in the middle part of A_1 are removed from B_1. Next, the holes which come off are moved to the middle part, among the two crossover points. To manage this, expect that the ends of the item are connected and start to push the items to the left from the second crossover point until the item which was originally before the second crossover point does not arrive to the position located before the first crossover point. Now, the middle part of A_1 can be copied to the realized free place at the middle of B_1. B_2 can be generated by the same process (see Fig. 4).

The cycle crossover (CX) does not use crossover points. Firstly, it copies the first item of A_1 to B_1, then it looks for the the first item of A_2 and searches it in A_1. It copies it to B_1. Next, it looks for the item at the same position in A_2, and it searches it again, and so on. It continues the process as long as it does not find an already copied item. By this time, it finished the cycle started from the first item of A_1. The remained empty fields of B_1 are filled with the items of A_2 from the same position (see Fig. 5).

From the classical one- and two-point crossovers the two-point crossover proved to be less efficient. Between the op-

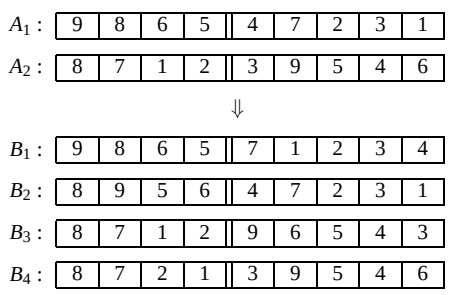

Fig. 3: One-point consistence preserver crossover

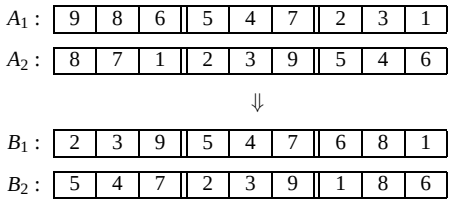

Fig. 4: Order Crossover (OX)

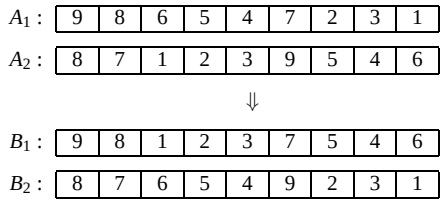

Fig. 5: Cycle Crossover (CX)

erators developed for permutations, OX and CX, the second one performed better in this problem, moreover the implementation of OX is much slower because of the lots of pushes, and at any chosen generation number the result of OX is worse either. In contrast with one-point crossover the CX was even faster. Concerning the convergence speed and the generation number, CX approximated the one-point crossover, what is more, in some cases CX overcame it. If not just the final result is watched, an interesting behavior of CX can be observed: at the beginning it ensures a much faster convergence speed than the one-point crossover, but later, CX's speed slows down, so the one-point crossover can make up its leeway (in generation number).

The role of the mutation in the GA is the assurance of the heuristic search, it tries to get to the individuals found in the, up to now, undiscovered part of the problem space.

The basic case of the consistence preserver mutation is the following: choose two points and a length in an individual, then swap the section of the randomized length from the first drawn point with the section of the same length starting from the second drawn point.

A more efficient solution is when not sections of a chosen length but sections with one item are swapped and it is repeated many times. According to our tests the second case gave much better convergence performance. A possible reason, which can confirm our result, is that this operator imitates mostly the attempts of a human timetable preparator. The timetable preparator also tries to swap some lessons on the chance of the elimination of a hard or a soft constraint.

The inversion, which is similar to mutation, was proved to be not an efficient operator in our case (accordant with the literatures of [4, 14, 3]).

A. *The penalty values*

The penalties indicate that in which rate the existence of a problem can make a timetable unusable. The unit of the uselessness is optional, so not really the penalty values itself, rather the ratio between the different aspects are important.

The biggest problems are caused by the class and the teacher clashes, and the finite room capacity, so these had the biggest penalty, uniformly 1000 points. The penalty for the violation of the teacher's strict availability was 500 points. Then the nonexistence of the multiple lessons and the existence of long breaks between two lessons come next in the importance order. A long break can be realized between two lessons or as the first lesson of a day. The second case in some situations does not always cause problems. During our test all the three problems got a penalty of 100 points.

Finally, the least penalized situations were the same type of sets' inequable distribution in a week, the high fluctuation of the lessons per day and the violation of teacher's soft availability.

The values of the penalties, and its ratios concerning to each other have less effect on the convergence speed. In accordance with its explanation they influence just the priority of the algorithm for which cases it optimizes more and for which cases less.

B. *Elitism*

The origination of a new generation from the actual generation is called a (bigger) step of the GA. The usage of overlapping population imitates mostly the inheritance found in the nature. In this case not the whole population is replaced by new individuals, but the parents and children live together in the next generation.

By using non-overlapping populations the old population is replaced completely with a new one. Generally, the non-overlapping populations give better results in the optimization problems, because it can check more new individuals, so it provides a better convergence speed. The marginal case of the overlapping and the non-overlapping population is the elitism. By using the elitism, the whole population is replaced with new individuals except the most viable, namely the elitist individual of the old population as it is kept in the new generation, too. The maintenance of the elite individual improves efficiently the deterministic search. Although [14] also declares that elitism improves the convergence speed, in our case it provided an unexpected growth in efficiency.

C. *The relation between the population size and the generation number*

The execution time of the algorithm is roughly proportional with the product of the population size and the generation number, but the convergence speed quite depends on the selection of the ratio between these. The small population can be processed during lot of generations, while on big populations just a few generation step can be made. In the big population case the selection and the crossover operators have great combinational possibilities to generate a new individual, while the small population size promotes the wandering of some promising ways of the problem space to find the details more deeply.

Our test runs showed that the optimal population size is approximately 20 individuals. If the population is smaller then the number of possible combinations is not big enough, but it is not worth of increasing the population size either as it is just a waste of time.

VI. EXPERIMENTAL RESULTS

Our tests were done on the real life data of a secondary school located in Budapest, Hungary, where the number of classes is 18 and the number of teachers is 67. There are 5 working days in a week, and on each day there are 7 lessons in average. Our genetic algorithm was optimized to these values, and finally our software produced at least comparable timetables to the used ones. The penalty values of the best individuals in the first generation were between 150000 and 200000, and after running of one million generations the values of the best individuals decreased below 2500. Furthermore, in the fittest timetable there was not any hard constraint violation. There were only some problems with

teacher's soft availability, with the bunchyness and with the equal distribution.

A. *Fine tuning of the probabilities of the mutation and of the crossover*

The mutation and the crossover probabilities control the stochastic and deterministic habits of the algorithm. Of course, these two things are not independent.

It is not expedient to choose extreme probability values. The probability value of mutation close to 0 makes our algorithm too sensible, it can easily stick in a local minimum. If it is close to 1 it overemphasizes the stochastic characteristic, it constantly goes away from the hardly approached vicinity of the minima, our algorithm transforms to random search. If the probability of the crossover is kept in a very low level, it oppresses the deterministic steps, and thus, it causes a slower convergence speed. Its high value does not allow the success for those advantageous gene parts which can show their advantages just in an individual come into being later.

The probability values of the crossover and of the mutation can also be changed during the optimization process. With the growth of the generation number the crossover probability is progressively decreased, while the mutation probability is increased [14]. Namely, at the beginning with the help of both, the crossover and the mutation operators, the problem space is mapped, the crossover operator can combine efficiently the optimal parts of the individuals' genetic substrate. However, after a while, our algorithm sticks in a local minimum and individuals with low penalty values coming from the environment of this minimum will dominate the population, the crossover operator will work on very similar individuals, instead of discovering new parts with lower penalty values. The progressive increment of the mutation probability tries to solve this problem with the decrement of the crossover probability at the same time. If the actual local minimum is the same as the global minimum, then the high value of the mutation probability is not convenient for us as it can damage our solution. With the elitism this can be avoided easily.

According to our tests, this general idea was usable in the timetabling problem, but it did not increase tremendously the convergence speed, however, if the penalty values are well tuned it decreased a bit the time needed for the optimization process.

Conforming to our test results, the optimal probability for the classical mutation was around 0.5, while for the one item mutation it was between 0.2 and 0.3, and the optimal number of items swapped at one time was between 3 and 5. These mutation probability values are much higher (especially in the first case) then in general, but it can be regarded as normal in this and in similar kind of problems [1]. A possible reason can be the tremendous problem space, as for the acceleration of the convergence speed the stochastic steps, the experiments have to be allowed to be effective. On the other hand the crossover cannot be implemented as freely as in the nature because of consistence preservation, in practice just half of the internal structure of a chromosome can be preserved.

The optimal value for the crossover probability was around 0.7. If the probability values change during the optimization process it is convenient to increase the mutation probability from 0.3 to 0.5 and to decrease the crossover probability from 0.7 to 0.5.

VII. CONCLUSIONS

A new representation, the set representation is introduced in this paper with which the real life situations of the educational institutions can be satisfied much better. Genetic Algorithms are used as an optimization method for our computationally NP-hard problem. For measuring how bad a timetable is several hard and soft constraints are defined. Different fitness functions and consistence preserver genetic operators are also tried. During our tests which are based on a secondary school's real life data different penalty values assigned to the different constraints, the population size and the generation number are optimized.

REFERENCES

[1] A. Álmos. *Scheduling algorithms for a video-dubbing studio*. INTCOM '98, Miskolc, Hungary, November 21-27, 1998.

[2] A. Álmos and A. R. Várkonyi-Kóczy. Genetic methods for point labeling on maps. In J. Vascak P. Sincak, editor, *Quo Vadis Computational Intelligence? (Studies in Fuzzyness and Soft Computing)*, Heidelberg, 2000. Physica-Verlag.

[3] D.L. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, 1989.

[4] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[5] Faragó A. and Fecher G. Genetic algorithms based timetabling for primary and secondary schools. Technical report, Technical University of Budapest, 1999.

[6] W. Junginger. Timetabling in germany – a survey. *Interfaces*, 16:66–74, 1986.

[7] aSc Applied Software Consultants. *aSc Timetables Version 2000, 2.2000.0.76*. http://www.asc.sk/rozvrhy.

[8] A. Abramson. Constructing schoool timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, 37:98–113, 1991.

[9] A. Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54:39–47, 1992.

[10] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.

[11] A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications Journal*, 1997.

[12] D. Beasley, D.R. Bull, and R.R. Martin. An overview of genetic algorithms. *University Computing*, 15(2):58–69, 1993.

[13] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans SMC*, 16:122–128, 1986.

[14] H-L. Fang. Investigating genetic algorithms for scheduling. Technical report, MSc Dissertation, Department of Artificial Intelligence, University of Edinburgh, 1992.