International Journal of Innovative
Computing, Information and Control
Volume **9**, Number **11**, November **2013**

ICIC International ©2013 ISSN 1349-4198
pp. 4565–4582

SELF-LEARNING GENETIC ALGORITHM FOR A TIMETABLING PROBLEM WITH FUZZY CONSTRAINTS

Radomir Perzina and Jaroslav Ramik

Centre of Excellence IT4Innovations
Division of the University of Ostrava
Institute for Research and Applications of Fuzzy Modeling
30. dubna 22, 701 03 Ostrava, Czech Republic
{ perzina; ramik }@opf.slu.cz

Received December 2012; revised April 2013

Abstract. *A Timetabling Problem is an NP-hard combinatorial optimization problem which lacks analytical solution methods. During the last two decades several algorithms have been proposed, most of which are based on heuristics like evolutionary computation methods. In this paper, to solve this problem we present a specific genetic algorithm with fuzzy constraints. Our method incorporates a self-learning genetic algorithm using indirect representation based on event priorities and heuristic local search operators to tackle real world timetabling problems. By using fuzzy sets we measure the violation of soft constraints in the fitness function in order to take care of inherent uncertainty and vagueness involved in real life data. The proposed technique satisfies all of the hard constraints of the problem and achieves a significantly better score in satisfying the soft constraints. The algorithm is computationally effective as it is demonstrated on a small, realistic example for which an optimal solution is already known due to exhaustive calculation. The structure of the algorithm enables parallel computations which are necessary for solving large real life problems.*
Keywords: Timetabling problem, Hard constraints, Soft constraints, Fuzzy sets

1. **Introduction.** In this paper, we deal with Timetabling Problem (TP), sometimes called the Course Timetable Problem (CTP), see [2,8,24,25]. This problem represents an important class of optimization problems in Operations Research. It is considered to be one of the most interesting practical problems faced by universities (schools, colleagues, professional education facilities, etc.) and education companies (offering, e.g., management skills courses) today. The problem can be formulated as the allocation of given resources (teachers, students, employees, etc. as well as classrooms) to objects (lectures, seminars, educational events, etc.) being placed in time space and satisfying all of the given constraints of existing facilities so that a set of desirable objectives is satisfied as nearly as possible. Here, we focus only on a so-called course (event, etc.) timetabling problem. The automation of timetabling problems is an important task as it saves a lot of work for people and institutions and provides optimal solutions that can boost productivity, quality of education and services. The real life timetabling problems have many different forms, such as education timetable (courses and examinations), employee timetable, timetable of sports events, timetable of construction, timetable of transportation, services. Timetabling problems and scheduling problems are generally NP-hard constrained optimization problems, see, e.g., [5], of a combinatorial nature. As a rule, no exact algorithm (excluding total enumeration) is known which generates a solution within a reasonable time. These problems are also classified as constraint satisfaction problems, see [5]. There are a number of versions of CTP differing from one area of application (e.g.,

university) to another (e.g., transportation company), see, e.g., [6,9]. Specific work has been done on this type of CTP problem in specific universities and many formulations and algorithms have been tried. One of the most important computing paradigms is the graph coloring concept, see [26], where vertices represent courses (or events) and an arc joins two vertices only if they cannot be scheduled at the same time. The problem here is to find the chromatic number of resulting graphs. The chromatic number problem is, however, also NP-hard. Due to the complexity of the problem, most of the publications concentrate on heuristic algorithms which try to find "good" approximate solutions, see [1-4,10,20-25,28]. Some of these include Genetic Algorithms (GA), see [6,13,27,30], Tabu Search (TS), see [11,16,32], and Simulated Annealing (SA), see [31]. Heuristic optimization methods are explicitly aimed at looking for good feasible solutions that eventually may not be optimal where the complexity of the problem or limited time available does not allow the finding of an exact solution. Generally, two questions arise:

(1) How fast is the "optimal" solution computed?

(2) How close is this solution to the exact optimal one?

Between time and quality a tradeoff is often required, and this is taken care of by running the simple algorithm more than once. The empirical evaluation of the heuristic method is based on the analytical difficulty involved in the problem and pathological worst case result. Recently, these heuristic tools have been combined among themselves with knowledge elements, see [13,23,31], as well as with more traditional approaches. Developing solutions with these tools offers two major advantages:

(i) Shorter development time than traditional approaches, and

(ii) Robust systems being insensitive to noisy and missing data.

Keeping in view recent achievements, this work attempts to develop genetic algorithms with fuzzy constraints for a CTP. A specific feature of our CT problem is a formulation of the soft constraints of the problem by using fuzzy sets, and in particular trapezoidal fuzzy numbers, allowing for the calculation of grades of membership. It is evaluated in the fitness function (i.e., the objective function) that characterizes the solution of the problem and calculates its overall fitness values as a sum of joint areas for all soft constraints. This formulation of the problem has been proven to be suitable for the reduction of calculation time. This is a significant and novel contribution of the paper because such formulation is very important from practical point of view in many CTP problems. Our method also incorporates an original formulation of the self-learning genetic algorithm using an indirect representation based on the event priorities and heuristic local search operators. We tested our approach on the real timetabling problem at Silesian University in Opava – School of Business Administration in Karvina, Czech Republic. The problem size is characterized by the following parameters: number of rooms is 43, number of events is 705, number of students is 1807, number of teachers is 112. We ran the genetic algorithm for that problem and it was terminated when there was no increase in fitness fiction for 1000 generations, which took about 5 hours on the standard model of PC with a 2 GHz processor. The final timetable satisfied all of the hard constraints, i.e., there were no time conflicts for teachers and all of the events were placed in acceptable rooms and also in acceptable times.

The contents of the paper are as follows. In Section 2, a CTP is described with corresponding nomenclature and notation. A specific new feature of the problem included in the overall fitness function is described in Section 3. Section 4 is devoted to a detailed formulation of the proposed genetic algorithm. An application of the GA in a case study of a smaller extent is presented in Section 5. The last sections summarize the results of the paper and give some hints for the future research in the area.

2. **Course Timetabling Problem.** A course timetabling problem (CTP) consists of finding the exact time allocation within a limited time period for a number of events (lectures and seminars) and assigning them to a number of resources (teachers, students and classrooms) so that the constraints are satisfied, see, e.g., [8,21]. The constraints to be satisfied by the timetable are usually divided into two categories: hard and soft ones. Hard constraints must be rigidly fulfilled. Such constraints include:

(i) No resource (teachers, student groups and rooms) may be assigned to different events (e.g., lectures) at the same time.

(ii) The rooms assigned to an event must belong to the set of valid resources for that event. In this regard, the event is held in a room if the proper infrastructural arrangements are there to organize the event, e.g., information technology.

(iii) The event is assigned to a teacher if he has the knowledge and capability for delivering that particular event.

On the other hand, it is desirable to fulfill soft constraints to the best extent possible, but it is not extremely essential for a feasible solution. Therefore, soft constraints can also be seen as optimization objectives for our algorithm:

(Objective 1) Scheduling an event within a particular window of the whole period (such as during evenings).

(Objective 2) Minimizing time gaps or travel times between adjacent lectures of the same teacher.

Now we describe the input data for the scheduling problem and formalize the optimization model. In this model we use the following notation:

- n_R – number of available rooms (classrooms, offices), $\{R_1, R_2, \ldots, R_{n_R}\}$ – set of available rooms.
- n_E – number of events (actions, lectures, seminars), $\{E_1,\ E_2,\ \ldots,\ E_{n_E}\}$ – set of events (actions).
- n_T – number of teachers, $\{T_1, T_2, \ldots, T_{n_T}\}$ – set of teachers.
- n_S – number of students (group of students), $\{S_1, S_2, \ldots, S_{n_S}\}$ – set of students (group of students).
- n_P – number of time slots (time periods), $\{P_1, P_2, \ldots, P_{n_P}\}$ – set of time slots (time periods).
- n_G – number of time-room slots, $\{G_1, G_2, \ldots, G_{n_G}\}$ – set of time-room slots.
- $C = \{c_{ij}\}$ – clash matrix with elements c_{ij}; $i = 1, 2, \ldots, n_E$; $j = 1, 2, \ldots, n_E$.
- $A = \{a_{ij}\}$ – room acceptance matrix with elements a_{ij}; $i = 1, 2, \ldots, n_R$; $j = 1, 2, \ldots, n_E$.

The purpose of the clash matrix C is to determine which events should not be scheduled at the same time for each event. Each element of the clash matrix c_{ij} is a binary variable, i.e., $c_{ij} \in \{0; 1\}$. If E_i and E_j cannot be scheduled at the same time, then $c_{ij} = 1$, otherwise, $c_{ij} = 0$. Clash elements are equal to one for all events that are performed by the same teacher or attended by the same student group. Some rooms are not acceptable for specific events given by the room acceptance matrix A. Each element of the room acceptance matrix a_{ij} is a binary variable, i.e., $a_{ij} \in \{0; 1\}$.

3. **Soft Constraints and Fitness Function.** Uncertainty measures and vagueness associated with the soft constraints of the problem in the final timetable schedule are modeled by fuzzy sets allowing grades of membership. Our approach allows for the expressing of some preferences to the ultimate schedule so that the related measure of violation is appropriately represented. Among the soft constraints we preferably consider the best availability schedule according to the wishes of the teacher.

Among a variety of shapes that can be used to represent membership functions we prefer rectangular, triangular and trapezoidal, and other shapes, e.g., bell curves, s-curves, are not applied here. Usually, a shape of membership function is subjective and allows for the expressing of a teacher's preferences, see Figure 1, showing an example of a trapezoidal membership function of a teacher.

A scheduled timetable is then expressed by a rectangular membership function $r(t)$ which is equal to 1 for all times when the teacher is doing the event. The scheduled timetable membership function $r(t)$ for a single event starting at time $u = 3$ and ending at time $v = 5$ is shown in Figure 2.

The membership function of satisfaction with teacher preferences $s(t)$ is then expressed by the minimum operator, i.e., by formula

$$s(t) = \min\{r(t), p(t)\}$$

The membership function of satisfaction with the teacher preferences $s(t)$ for $r(t)$ and $p(t)$ above is shown in Figure 3.

The teacher's satisfaction with the scheduled timetable $H(r, p)$ is based on the joint area under the membership functions s and r using the formula

$$H(r, p) = 1 - \frac{\int_0^\infty s(t)dt}{\int_0^\infty r(t)dt} \tag{1}$$

where calculation of $\int_0^\infty s(t)dt$ depends on the values of parameters a, b, c, d, and $\int_0^\infty r(t)dt$ on the values of u, v as shown in Table 1.

After the timetabler has produced a timetable, it is evaluated by the fitness function which characterizes the solution and calculates its overall fitness values as a sum of weighted scores and penalties for all constraints, hard and soft. The fitness function z is

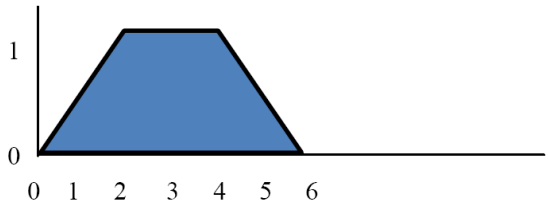

FIGURE 1. Teacher preference membership function $p(t)$

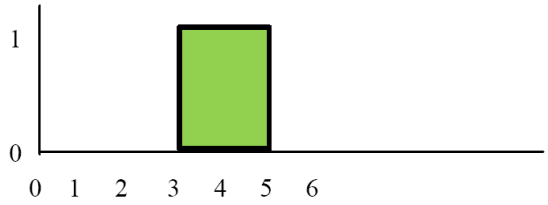

FIGURE 2. Scheduled timetable membership function $r(t)$

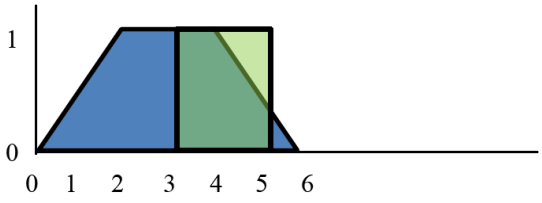

FIGURE 3. Membership function of satisfaction with teacher preferences $s(t)$

TABLE 1. Calculation of joint area S

$u < a$	$v < a$	$S = 0$
	$a \le v < b$	$S = \frac{(v-a)^2}{2(b-a)}$
	$b \le v < c$	$S = \frac{2v-a-b}{2}$
	$c \le v < d$	$S = \frac{d-a+c-b}{2} - \frac{(d-v)^2}{2(d-c)}$
	$v \ge d$	$S = \frac{d-a+c-b}{2}$
$a \le u < b$	$a \le v < b$	$S = \frac{(v-a)^2-(u-a)^2}{2(b-a)}$
	$b \le v < c$	$S = \frac{2v-a-b}{2} - \frac{(u-a)^2}{2(b-a)}$
	$c \le v < d$	$S = \frac{2v-a-b}{2} - \frac{(u-a)^2}{2(b-a)} - \frac{(d-v)^2}{2(d-c)}$
	$v \ge d$	$S = \frac{d-a+c-b}{2} - \frac{(u-a)^2}{2(b-a)}$
$b \le u < c$	$b \le v < c$	$S = v - u$
	$c \le v < d$	$S = \frac{c+d-2u}{2} - \frac{(d-v)^2}{2(d-c)}$
	$v \ge d$	$S = \frac{c+d-2u}{2}$
$c \le u < d$	$c \le v < d$	$S = \frac{(d-u)^2-(d-v)^2}{2(d-c)}$
	$v \ge d$	$S = \frac{(d-v)^2}{2(d-c)}$
$u \ge d$	$v \ge d$	$S = 0$

defined in (2), and the optimization problem is defined as follows:

$$
\begin{aligned}
& z = \sum_{i=1}^{n_T} H_i\,(r,p) \to \min; \qquad (2)\\
\text{s.t. } & \sum_{i=1}^{n_E} x_{ijk} \le 1,\ j = 1,2,\ldots,n_R,\ k = 1,2,\ldots,n_P,\\
& \sum_{j=1}^{n_R}\sum_{k=1}^{n_P} x_{ijk} = 1,\ i = 1,2,\ldots,n_E,\\
& \sum_{j=1}^{n_R}\sum_{k=1}^{n_P} a_{jk}\cdot x_{ijk} = 1,\ i = 1,2,\ldots,n_E,\\
& \sum_{i=1}^{n_E}\sum_{l=1}^{n_E}\sum_{j}^{n_R}\sum_{k}^{n_P}\sum_{m}^{n_R}\sum_{n}^{n_P} c_{il} sametime\,(x_{ijk}, x_{lmn}) = 0,
\end{aligned}
$$

where x_{ijk} is a binary variable determining whether the event E_i is scheduled in the room R_j and time-slot P_k. The expression $sametime(x_{ijk},\ x_{lmn})$ is the function that is equal

to 1 if the event E_i is scheduled at the same time as the event E_l, otherwise it is equal to zero, i.e.,

$$sametime(x_{ijk}, x_{lmn}) = \begin{cases} 1 & \text{for } k = n, \\ 0 & \text{otherwise.} \end{cases}$$

4. **Genetic Algorithm for Timetabling Problem.** In this section, we present a genetic algorithm (GA) for our course timetabling problem. To solve the timetabling problem, we develop an optimization method based on GA that incorporates a number of techniques and specific local search operators. It is well known that GA is an iterative search procedure widely used in solving optimization problems motivated by biological models of evolution. A population of candidate solution is maintained in each iteration, see, e.g., [12]. Specific genetic operators such as mutation and crossover are applied to evolve solutions and find a "good solution" that has a high chance to survive for the next iteration. Firstly, the method is required to encode the timetable solution into an encoded form or chromosome suitable for applying genetic operators. Generally, two different approaches are considered, which are *direct* and *indirect*. A *direct* representation directly encodes all event attributes: classroom, teacher, time slot, etc. for all events. GA has to make decisions for all of the timetable parameters and deliver a complete and constraint free schedule. However, directly encoded solutions that undergo genetic operators frequently result in invalid solutions that have to be handled. An *indirect* representation [24], on other hand, considers an encoded solution, i.e., a chromosome that usually represents an ordered list of events which are placed into a timetable according to some predefined method (*timetable builder*).

4.1. **Timetable builder.** The timetable builder can use any combination of heuristics and local searches to place events into a timetable while observing the constraints of the problem. For GA implementation of this work, we have considered an indirect representation that encodes 3 fields for each event into the chromosome:

(a) Teachers to be assigned to events – here, this problem is performed beforehand (manually) as each teacher's profession (specialization) determines his/her allocation to particular events.

(b) Classroom where the event will be held.

(c) Time slot within the classroom. All fields are first encoded and then entered into the chromosome as real numbers. When GA produces such a solution, it first decodes it to gain these 3 fields for every event in the schedule. Then the timetabler is invoked and works as follows:

(i) It sorts events according to their priority values in ascending order. Values with high priorities are placed first.

(ii) It takes the first event with the highest priority, marks it as taken, and places it into the schedule.

(iii) Starting from a time slot, it places the event and checks if any constraint is violated. If the allocation is not fixed, the algorithm moves on to the next event.

(iv) The violated constraints are allocated to the event into subsequent time periods until all constraints are satisfied.

(v) If no time period exists for which all constraints are satisfied, the event is marked to violate the maximum time periods which exceed the constraints.

(vi) The algorithm continues with the next event in the list. When all events have been processed, the timetabler stops. The timetabler satisfies all hard constraints and all other constraints are satisfied in a satisfaction degree calculated by the fitness function.

4.2. **Genetic algorithm encoding.** Encoding is a major element of every genetic algorithm. The structure of the self-adaptive genetic algorithm's encoding is depicted in Figure 4. The idea of the proposed encoding consists of a redundancy of information through a hierarchical evaluation of individuals.

As we can see, in the population each individual – chromosome is composed of N_g genes, where each gene is linked to exactly one optimized variable. Each gene is composed of N_e gene elements. The number of gene elements varies for each gene, and this is continuously updated throughout evolution. Each gene element contains low-level parameters which encode the optimized variables and parameters of evolution. All parameters are listed in Table 2.

The upper index "^{E}" denotes that it is a gene element value of the parameter. As the encoding is hierarchical, there are several levels of the parameters, so gene values of

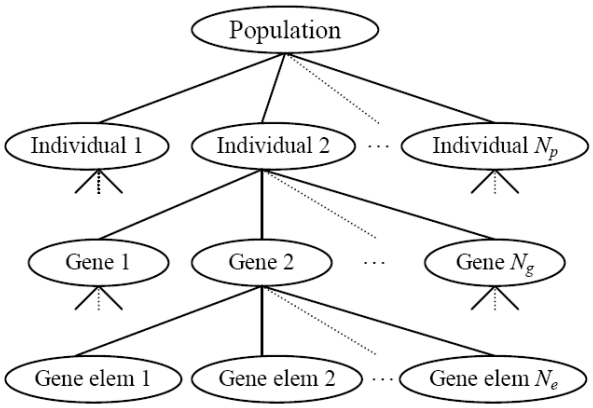

FIGURE 4. The structure of population

TABLE 2. The structure of a gene element

Name	Description	Range
x^E	Optimized variable	$[0; 1]$
q_m^E	Parameter of mutation	$[-1; 1]$
r_m^E	Radius of mutation	$[0; 0.5]$
p_c^E	Probability of crossover	$[0; 1]$
r_c^E	Ratio of crossover	$[0; 1]$
q_d^E	Parameter of deletion	$[-0.1; 0.1]$
q_u^E	Parameter of duplication	$[-0.1; 0.1]$
s_m^E	Identifier of myself for mating	$[0; 1]$
s_w^E	Wanted partner for mating	$[0; 1]$
r_r^E	Ratio of replacement	$[0; 1]$
r_t^E	Ratio of population for selection	$[0; 1]$
r_p^E	Ratio of population for 2$^{\text{nd}}$ partner selection	$[0; 1]$
c_d^E	Coefficient of death	$[0; 1]$
N_p^E	Wanted size of population	$[0; 1]$

parameters are marked by the upper index "G", individual values by "I" and population values by "P". Since genetic operators are applied only to the low level values of parameters (gene element), the upper level values of parameters cannot be updated directly through the evolution process, only indirectly by the evaluation mechanism from low level values.

4.3. **Gene parameters evaluation.** The encoding is polyploiditial, i.e., each gene is composed of N_e gene elements. The number of gene elements is variable and undergoes evolution. Gene values of gene elements are evaluated by Formula (3) as

$$X^G = \frac{1}{N_e} \sum_{i=1}^{N_e} X_i^E \tag{3}$$

where X stands for parameters that must be evaluated, i.e., x, s_m, s_w, r_r, r_t, r_p, c_d, N_p, i_t.

4.4. **Individual parameters evaluation.** Parameters concerning the whole individual, such as s_m^I, s_w^I, r_r^I, r_t^I, r_p^I, c_d^I, N_p^I are evaluated by Formula (4) as

$$X^I = \frac{1}{N_g} \sum_{i=1}^{N_g} X_i^G. \tag{4}$$

The number of genes N_g is not variable, because one gene contains exactly one optimized variable.

4.5. **Population parameters evaluation.** Parameters concerning the whole population, r_r^P, r_t^P, c_d^P, N_p^P, are evaluated as weighted average, with weights according to their *relative fitness* w_f defined as

$$w_f = \frac{N_p^P - i + 1}{\frac{\left(1+N_p^P\right)N_p^P}{2}}, \tag{5}$$

where i is an index of i^{th} individual in population sorted by fitness in descending order, i.e., the individual with the highest value of the fitness function has the value of i equal to 1, and the individual with the second highest value of the fitness function has the value of i equal to 2, etc.

4.6. **Genetic operators.** A standard genetic algorithm uses standard genetic operators:

(1) Selection – a standard tournament selection method.
(2) Crossover – a standard uniform crossover with probability of crossover p_c (e.g., $p_c = 0.5$).
(3) Mutation – a standard mutation operation with probability p_m (e.g., $p_m = 0.05$).

As the proposed encoding is specific, the genetic operators must be adjusted to fit the encoding. There are not only common genetic operators such as selection, crossover or mutation used, but also some specific ones, as described in following text.

4.6.1. *Selection.* In genetic algorithms the selection of both mating parents is usually based on their fitness, but this is not true in nature. In nature, a winner of a tournament selects his/her partner according to his/her individual preferences. One important thing is that he/she cannot take into account his genotype, i.e., directly the values of his/her genes nor his/her fitness, but only his/her phenotype, i.e., only expression of the genes to the outside. In a similar way, we try to imitate nature by using parameters s_m^I and s_w^I. The parameter s_w^I represents individual preferences for mating, and the parameter s_m^I represents an individual's phenotype for mating, so the first parent is selected by

a tournament selection method with a variable ratio of population r_t^P from which the fittest individual is selected. The second parent is selected according to the individual's preferences represented by the parameter s_w^I, i.e., the first parent chooses an individual with the minimal value of expression $\left|s_w^I - s_m^I\right|$, but selection is made from only a limited ratio of population r_p^I.

4.6.2. *Crossover.* The crossover operator is applied to every gene element of the first parent with the probability p_c^E. The crossover itself proceeds only between gene elements of mating parents according to formula

$$X_3^E = X_1^E + \left(X_2^E - X_1^E\right) \cdot r_c^E \tag{6}$$

where X stands for all parameters of a gene element, see Table 2, r_c^E is a ratio of crossover of the first parent defined in this gene element, the lower index "$_1$" denotes the gene element of the first parent, the index "$_2$" denotes the second parent and the index "$_3$" denotes the child of both parents. The gene element of the second parent is selected randomly, but it is of the same gene as the gene element of the first parent.

4.6.3. *Mutation.* The mutation operator is applied to every gene element with probability $p_m^E = \left|q_m^E\right|$. Notice that the probability of mutation is calculated as the absolute value of the parameter of mutation, because the mean value of p_m^E should be zero. Moreover, every gene element has its own probability of mutation. The mutation formula is defined as

$$X_{new}^E = X_{old}^E + \left(X_{\max}^E - X_{\min}^E\right) \cdot U\left(-r_m^E, r_m^E\right) \tag{7}$$

where X stands for all parameters of the gene element, $U(a, b)$ is a random variable with uniform probability distribution in the interval $[a; b]$, X_{new}^E is the value of the parameter after mutation, X_{old}^E is the original value of the parameter, and $X_{\max}^E$ ($X_{\min}^E$) is the maximal (minimal) allowed bit element value of the parameter in Table 2.

4.6.4. *Duplication.* The duplication operator is applied to every gene element with probability $p_u^E = \left|q_u^E\right|$. The gene element is duplicated (copied) with the same value of all parameters, with the only exception being that the values of parameter q_u^E of both gene elements are divided by the coefficient 2 in order to inhibit the exponential growth of bit elements.

4.6.5. *Deletion.* The deletion operator is applied to every gene element with probability $p_d^E = \left|q_d^E\right|$. It means that the gene element is removed from the gene. By using deletion and duplication operators the degree of polyploidity is controlled.

4.6.6. *Replacement of individuals.* For every individual the parameter of a life strength – L is defined. When the individual is created, its life strength L is set to one, and in every generation it is multiplied by the coefficient c_L defined as

$$c_L = 1 - c_d^P \left(1 - w_f\right). \tag{8}$$

Evidently, through evolution a less fit individual causes a greater decrease in L. In every generation all X^P parameters are evaluated, and by using the above listed genetic operators $N_p^P \cdot r_r^P$ new individuals are created. Then a randomly selected individual is "killed" with probability $(1 - L)$. This process of eliminating individuals is repeated until only N_p^P individuals survive in the population.

5. **Case Study – Computational Results.** In this section, we present a realistic example – case study with a relatively small extent so that the exact optimal solution can be calculated by total enumeration of 12 (about 500 million) variants of feasible timetables.

Let us suppose we have 10 events (lectures or seminars) $E_1, E_2, \ldots, E_{10}$ and 3 teachers T_1, T_2 and T_3. Teacher T_1 is teaching events E_1, E_2, E_3, teacher T_2 is teaching events E_4, E_5, E_6 and teacher T_3 is teaching events E_7, E_8, E_9, E_{10}. The clash matrix for such a schedule is shown in Table 3.

Now, suppose there are four students (or groups of students) S_1, S_2, S_3 and S_4. Student S_1 is attending events E_1, E_3, E_5, E_{10}, student S_2 is attending events E_2, E_3, E_6, E_8, student S_3 is attending events E_1, E_4, E_5, E_9 and student S_4 is attending events E_2, E_4, E_7, E_{10}. A sample clash matrix for such schedule is shown in Table 4.

The clash matrix for students and teachers is then calculated as

$$c_{ij} = \max\left\{c_{ij}^T, c_{ij}^S\right\},$$

where c_{ij}^T is a clash matrix element for teachers and c_{ij}^S is a clash matrix element for students. The total clash matrix for such a schedule is shown in Table 5.

The purpose of the room acceptance matrix A is to determine which events can be scheduled in particular rooms. Each element of the clash matrix a_{ij} is a binary variable $a_{ij} \in \{0; 1\}$. If $a_{ij} = 1$, then event E_j can be scheduled in the room R_i. Let us suppose we have two rooms R_1 and R_2. Events E_1, E_2, E_3, E_4, E_5, E_6 can be scheduled in room R_1

TABLE 3. Clash matrix for teachers

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
E_1	0	1	1	0	0	0	0	0	0	0
E_2	1	0	1	0	0	0	0	0	0	0
E_3	1	1	0	0	0	0	0	0	0	0
E_4	0	0	0	0	1	1	0	0	0	0
E_5	0	0	0	1	0	1	0	0	0	0
E_6	0	0	0	1	1	0	0	0	0	0
E_7	0	0	0	0	0	0	0	1	1	1
E_8	0	0	0	0	0	0	1	0	1	1
E_9	0	0	0	0	0	0	1	1	0	1
E_{10}	0	0	0	0	0	0	1	1	1	0

TABLE 4. Clash matrix for students

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
E_1	0	0	1	1	1	0	0	0	1	1
E_2	0	0	1	1	0	1	1	1	0	1
E_3	1	1	0	0	1	1	0	1	0	1
E_4	1	1	0	0	1	0	1	0	1	1
E_5	1	0	1	1	0	0	0	0	1	1
E_6	0	1	1	0	0	0	0	1	0	0
E_7	0	1	0	1	0	0	0	0	0	1
E_8	0	1	1	0	0	1	0	0	0	0
E_9	1	0	0	1	1	0	0	0	0	0
E_{10}	1	1	1	1	1	0	1	0	0	0

TABLE 5. Clash matrix for teachers and students

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
E_1	0	1	1	1	1	0	0	0	1	1
E_2	1	0	1	1	0	1	1	1	0	1
E_3	1	1	0	0	1	1	0	1	0	1
E_4	1	1	0	0	1	1	1	0	1	1
E_5	1	0	1	1	0	1	0	0	1	1
E_6	0	1	1	1	1	0	0	1	0	0
E_7	0	1	0	1	0	0	0	1	1	1
E_8	0	1	1	0	0	1	1	0	1	1
E_9	1	0	0	1	1	0	1	1	0	1
E_{10}	1	1	1	1	1	0	1	1	1	0

and E_4, E_5, E_6, E_7, E_8, E_9, E_{10} can be scheduled in room R_2. A sample room acceptance matrix for such a schedule is shown in Table 6.

TABLE 6. Room acceptance matrix

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
R_1	1	1	1	1	1	1	0	0	0	0
R_2	0	0	0	1	1	1	1	1	1	1

In order to demonstrate the calculation of the objective function, let us suppose we have teacher time preferences as described in Table 7, and time-room slots are assigned to events as shown in Table 8.

TABLE 7. Teacher time preferences

	a	b	c	d
T_1	0	0	3	4
T_2	1	2	3	4
T_3	2	3	5	6

Then the ratio of satisfaction for T_1 is 0, the ratio of satisfaction for T_2 is 0.833, and the ratio of satisfaction for T_3 is 0.375. Therefore, the total value of the objective function $z = 1.208$.

5.1. **Decoding chromosome.** We have 10 events and 12 time-room slots so the chromosome for such a simple timetable will have 22 genes. Let us suppose that after evaluation the gene values of parameters are as shown in Table 9.

First, events must be sorted according to the values of their parameters in the chromosome (part A), and as a result the order is E_{10}, E_2, E_3, E_6, E_1, E_9, E_5, E_4, E_7, E_8. The time-room slots have to be sorted according to the values of their parameters in the part B of the chromosome, so the order will be G_7, G_8, G_4, G_{10}, G_{12}, G_9, G_6, G_5, G_1, G_2, G_3, G_{11}. The process of assigning time-room slots is depicted in Table 10. The selected time-room slot for each event is marked by the symbol **X**, and time-room slots used for previous events have a grey background.

In the first step, we first assigned time-room slot G_7 to the first event, E_{10}. In the second step we should assign G_8 to E_2, however G_8 corresponds to room R_2 which is

TABLE 8. Time-room slot assignments

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
R_1M_1	1	0	0	0	0	0	0	0	0	0
R_1M_2	0	1	0	0	0	0	0	0	0	0
R_1M_3	0	0	1	0	0	0	0	0	0	0
R_1M_4	0	0	0	1	0	0	0	0	0	0
R_1M_5	0	0	0	0	0	0	0	0	0	0
R_1M_6	0	0	0	0	1	0	0	0	0	0
R_2M_1	0	0	0	0	0	1	0	0	0	0
R_2M_2	0	0	0	0	0	0	0	0	1	0
R_2M_3	0	0	0	0	0	0	0	0	0	0
R_2M_4	0	0	0	0	0	0	0	1	0	0
R_2M_5	0	0	0	0	0	0	0	0	0	1
R_2M_6	0	0	0	0	0	0	1	0	0	0

TABLE 9. Chromosome

Part	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.45	0.13	0.22	0.87	0.74	0.36	0.88	0.99	0.67	0.01	0.54	0.63	0.71	0.31	0.48	0.45	0.14	0.18	0.42	0.37	0.94	0.39

TABLE 10. Building timetable

Step		G_7	G_8	G_4	G_{10}	G_{12}	G_9	G_6	G_5	G_1	G_2	G_3	G_{11}
1.	E_{10}	**X**											
2.	E_2			**X**									
3.	E_3							**X**					
4.	E_6		**X**										
5.	E_1								**X**				
6.	E_9				**X**								
7.	E_5					**X**							
8.	E_4						**X**						
9.	E_7												**X**
10.	E_8												

not suitable for E_2, therefore the next suitable time-room slot must be selected, i.e., G_4. In the third step, we should first assign unused time-room slot G_8 to E_3, but again, G_8 corresponds to room R_2 which is not suitable for E_3, so the first suitable time-room slot for E_3 is G_6. In the fourth step we can finally assign the skipped time-room slot G_8 to E_6. We continue this way until either all of the events are assigned or any remaining events cannot be assigned to a suitable time-room slot. In this example, the last event E_8 cannot be assigned to any suitable time room slot, and as a result the high penalty coefficient will be added to the objective Function (2). The final timetable according to the chromosome provided in the example will be: $E_1 \to G_5$, $E_2 \to G_4$, $E_3 \to G_6$, $E_4 \to G_9$, $E_5 \to G_{12}$, $E_6 \to G_8$, $E_7 \to G_{11}$, $E_8 \to$*Unassigned*, $E_9 \to G_{10}$, $E_{10} \to G_7$, where symbol $\to$ stands for "is assigned to".

5.2. **Evolution process of the standard GA.** Firstly, we will describe standard evolution using a standard genetic algorithm. We start with an initial population of randomly

generated chromosomes. Let us suppose that we have the following 4 chromosomes in the initial population:

X1	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.96	0.78	0.46	0.45	0.13	0.15	0.63	0.58	0.93	0.48	0.02	0.98	0.85	0.80	0.33	0.61	0.27	0.34	0.42	0.31	0.35	0.82

X2	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.73	0.10	0.02	0.36	0.31	0.94	0.51	0.36	0.85	0.05	0.60	0.63	0.77	0.10	0.59	0.01	0.38	0.43	0.65	0.30	0.47	0.02

X3	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.56	0.94	0.00	0.59	0.81	0.22	0.21	0.87	0.55	0.20	0.16	0.65	0.62	0.86	0.84	0.72	0.05	0.33	0.81	0.41	0.51	0.84

X4	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.24	0.61	0.45	0.88	0.91	0.86	0.57	0.98	0.57	0.79	0.77	0.82	0.19	0.46	0.66	0.63	0.92	0.22	0.52	0.91	0.33	0.39

The fitness function is then calculated for each chromosome in the initial population as follows:

Individual	**Fitness function**
X1	−60002.083
X2	−40002.083
X3	−40001.625
X4	−60001.542

The first applied operator is a tournament selection. Let us suppose 4 pairs of chromosomes were selected in tournaments: (X1, X2), (X2, X3), (X3, X4), (X2, X3). A crossover operator is then applied to each pair of chromosomes with probability $p_c = 0.5$ (crossed over genes are highlighted). The new value of each gene is calculated as the arithmetic average of the parents' gene values. After crossover we get 4 new offspring X1', X2', X3', X4':

X1'	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.85	0.78	0.24	0.45	0.22	0.55	0.63	0.58	0.89	0.27	0.31	0.98	0.81	0.45	0.33	0.61	0.27	0.39	0.54	0.31	0.41	0.42

X2'	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.73	0.10	0.02	0.48	0.31	0.58	0.36	0.62	0.70	0.13	0.60	0.64	0.77	0.48	0.59	0.37	0.22	0.43	0.65	0.36	0.49	0.02

X3'	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.56	0.78	0.23	0.74	0.86	0.22	0.21	0.87	0.55	0.50	0.47	0.65	0.41	0.86	0.84	0.68	0.05	0.28	0.67	0.66	0.51	0.62

X4'	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.73	0.52	0.01	0.48	0.56	0.58	0.36	0.62	0.70	0.13	0.38	0.64	0.70	0.48	0.72	0.37	0.22	0.38	0.73	0.36	0.49	0.43

The last applied operator is a mutation, which adds a random value within an interval $[-0.1; 0.1]$ to each gene value with probability $p_m = 0.5$ to each offspring (mutated genes are highlighted). If the gene value after mutation is less than 0 or greater than 1, then

the new value of the gene is set to 0 or 1. After mutation we get 4 new chromosomes X1", X2", X3", X4":

X1"	A										B											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.92	0.70	0.24	0.45	0.22	0.55	0.63	0.58	0.93	0.27	0.22	1.00	0.81	0.45	0.24	0.61	0.25	0.35	0.47	0.31	0.33	0.42

X2"	A										B											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.75	0.03	0.07	0.48	0.41	0.63	0.39	0.58	0.64	0.13	0.60	0.64	0.86	0.48	0.56	0.32	0.22	0.41	0.65	0.42	0.44	0.02

X3"	A										B											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.56	0.78	0.23	0.72	0.86	0.22	0.21	0.87	0.55	0.50	0.52	0.65	0.46	0.84	0.78	0.68	0.13	0.28	0.67	0.74	0.55	0.62

X4"	A										B											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.64	0.45	0.01	0.48	0.56	0.54	0.36	0.64	0.70	0.04	0.38	0.64	0.65	0.48	0.65	0.37	0.22	0.38	0.78	0.38	0.52	0.43

Now each chromosome in the new population is evaluated by a fitness function and a new generation is again created using the genetic operators. The evolution process continues until there is no increase in fitness function for 100 generations. The solution of the problem is the fittest individual from all generations with a fitness function value of 0.70833:

X1*	A										B											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Value	0.91	0.18	0.23	0.45	0.51	0.62	0.63	0.78	0.83	0.89	0.09	0.89	0.05	0.11	0.96	0.98	0.40	0.31	0.26	0.56	0.84	0.73

This chromosome can be decoded into the timetable described in Table 11.

TABLE 11. Final timetable

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
R_1M_1	0	0	**1**	0	0	0	0	0	0	0
R_1M_2	**1**	0	0	0	0	0	0	0	0	0
R_1M_3	0	**0**	0	0	0	0	0	0	0	0
R_1M_4	0	0	0	**1**	0	0	0	0	0	0
R_1M_5	0	0	0	0	0	0	0	0	0	0
R_1M_6	0	0	0	0	0	0	0	0	0	0
R_2M_1	0	0	0	0	0	0	**1**	0	0	0
R_2M_2	0	0	0	0	0	**1**	0	0	0	0
R_2M_3	0	0	0	0	**1**	0	0	0	0	0
R_2M_4	0	0	0	0	0	0	0	**1**	0	0
R_2M_5	0	0	0	0	0	0	0	0	0	**1**
R_2M_6	0	0	0	0	0	0	0	0	**1**	0

5.3. **Evolution process of the self-adaptive GA.** The structure of the proposed genetic algorithm is complex, and therefore it would be very confusing trying to demonstrate the evolution cycle with all of the parameters (there would be over 600 parameters in a single chromosome for this optimization problem). An example of such a chromosome is shown in Table 12.

TABLE 12. Chromosome of the self-adaptive GA

X1	*A*										*B*											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Val 1	**0.96**	**0.78**	**0.46**	**0.45**	**0.13**	**0.15**	**0.63**	**0.58**	**0.93**	**0.48**	**0.02**	**0.98**	**0.85**	**0.80**	**0.33**	**0.61**	**0.27**	**0.34**	**0.42**	**0.31**	**0.35**	**0.82**
q_m^E	0.85	0.16	0.63	0.54	0.48	0.98	0.89	0.46	0.93	1.00	0.49	0.85	0.76	0.04	0.94	0.75	0.54	0.09	0.63	0.66	0.48	0.78
r_m^E	0.87	0.94	0.95	0.96	0.90	0.42	0.12	0.88	0.04	0.65	1.00	0.08	0.40	0.62	0.88	0.83	0.86	0.43	0.27	0.61	0.66	0.12
p_c^E	0.56	0.78	0.55	0.67	0.62	0.19	0.46	0.78	0.12	0.81	0.08	0.26	0.09	0.68	1.00	0.72	0.72	0.13	0.65	0.75	0.61	0.77
r_c^E	0.66	0.84	0.63	0.42	0.41	0.16	0.65	0.81	0.80	0.27	0.19	0.06	0.76	0.43	0.72	0.08	0.82	0.35	0.19	0.27	0.60	0.88
q_d^E	0.22	0.43	0.76	0.03	0.60	0.18	0.77	0.84	0.59	0.02	0.07	0.12	0.69	0.37	0.09	0.90	0.08	0.59	0.34	0.64	0.95	0.01
q_u^E	0.27	0.33	0.44	0.86	0.16	0.69	0.07	0.57	0.61	0.91	0.67	0.32	0.40	0.91	0.39	0.06	0.02	0.28	0.45	0.71	0.18	0.58
s_m^E	0.29	0.83	0.62	0.47	0.31	0.41	0.75	0.84	0.98	0.17	0.49	0.65	0.86	0.19	0.27	0.63	0.03	0.63	0.21	0.11	0.15	0.35
s_w^E	0.45	0.75	0.08	0.61	0.17	0.38	0.99	0.33	0.77	1.00	0.57	0.96	0.64	0.27	0.46	0.30	0.12	0.54	0.40	0.49	0.54	0.65
r_r^E	0.57	0.30	0.83	0.17	0.18	0.88	0.53	0.35	0.20	0.23	0.43	0.67	0.58	0.44	0.68	0.56	0.23	0.78	0.27	0.46	0.64	0.48
r_t^E	0.68	0.10	0.50	0.99	0.72	0.50	0.53	0.89	0.15	0.14	0.65	0.01	0.24	0.21	0.01	0.94	0.39	0.41	0.04	0.89	0.25	0.85
r_p^E	0.85	0.30	0.24	0.66	0.60	0.98	0.65	0.38	0.13	0.53	1.00	0.10	0.21	0.10	1.00	0.96	0.85	1.00	0.45	0.82	0.76	0.11
c_d^E	0.35	0.42	0.65	0.99	0.96	0.18	0.36	0.93	0.74	0.82	0.28	0.38	0.01	0.18	0.51	0.72	0.27	0.86	0.92	0.09	0.21	0.78
N_p^E	0.30	0.07	0.91	0.25	0.28	0.16	0.12	0.89	0.58	0.64	0.83	0.23	0.04	0.49	0.79	0.25	0.37	0.58	0.66	0.16	0.45	0.27
Val 2	**0.21**	**0.17**	**0.57**	**0.02**	**0.53**	**0.26**	**0.75**	**0.77**	**0.75**	**0.78**	**0.59**	**0.31**	**0.50**	**0.36**	**0.17**	**0.30**	**0.10**	**0.86**	**0.99**	**0.11**	**0.90**	**0.71**
q_m^E	0.61	0.28	0.23	0.75	0.48	0.37	0.83	0.63	0.83	0.31	0.40	0.41	0.41	0.24	0.09	0.04	0.77	0.13	0.87	0.50	0.09	0.80
r_m^E	0.32	0.72	0.75	0.57	0.17	0.98	0.63	0.45	0.32	0.01	0.80	0.10	0.95	0.21	0.75	0.13	0.14	0.49	0.52	0.26	0.15	0.11
p_c^E	0.17	0.68	0.02	0.17	0.87	0.60	0.29	0.83	0.07	0.38	0.72	0.77	0.18	0.39	0.96	0.81	0.25	0.76	0.69	0.73	0.34	0.92
r_c^E	0.45	0.95	0.33	0.63	0.69	0.08	0.08	0.86	0.18	0.96	0.65	0.41	0.09	0.24	0.32	0.40	0.51	0.47	0.65	0.99	0.26	0.21
q_d^E	0.15	0.27	0.33	0.99	0.94	0.81	0.77	0.37	0.31	0.28	0.63	0.33	0.39	0.76	0.24	0.42	0.60	0.21	0.16	0.68	0.27	0.69
q_u^E	0.63	0.83	0.60	0.12	0.70	0.68	0.04	0.82	0.96	0.78	0.67	0.61	0.39	0.04	0.56	0.30	0.57	0.29	0.69	0.40	0.53	0.71
s_m^E	0.04	0.11	0.98	0.67	0.69	0.09	0.82	0.52	0.16	0.70	0.38	0.33	0.16	0.95	0.11	0.95	0.34	1.00	0.14	0.59	0.52	0.05
s_w^E	0.05	0.75	0.32	0.81	0.14	0.17	0.15	0.68	0.27	0.02	0.89	0.99	0.90	0.68	0.29	0.45	0.32	0.72	0.63	0.70	0.10	0.36
r_r^E	0.09	0.38	0.27	0.31	0.63	0.51	0.51	0.98	0.42	0.73	0.42	0.25	0.56	0.74	0.71	0.96	0.31	0.18	0.28	0.53	0.28	0.43
r_t^E	0.63	0.70	0.25	0.86	0.07	0.03	0.97	0.72	0.60	0.01	0.30	0.80	0.68	0.58	0.43	1.00	0.04	0.67	0.73	0.76	0.28	0.18
r_p^E	0.01	0.19	0.52	0.39	0.89	0.06	0.04	0.94	0.22	0.94	0.03	0.76	0.49	0.85	0.88	0.97	0.31	0.30	0.20	0.56	0.74	0.26
c_d^E	0.47	0.29	0.95	0.83	0.02	0.77	0.10	0.34	0.43	0.93	1.00	0.23	0.71	0.84	0.68	0.02	0.64	0.65	0.17	0.38	0.41	0.10
N_p^E	0.03	0.89	0.07	0.42	0.60	0.59	0.98	0.01	0.03	0.63	0.84	0.90	0.35	0.82	0.56	0.75	0.06	0.63	0.58	0.72	0.84	0.68

Therefore, for the purpose of this demonstration we have simplified the structure of the genetic algorithm so that all of the main principles remain and the size of the chromosome is still manageable. Here, each gene has just a single gene element, and each gene element contains just 2 parameters: one parameter for optimized variable and one control parameter (radius of mutation). From all of the defined genetic operators, only mutation is used. All of the other control parameters and genetic operators are based on the same principle.

Let us suppose that we have the following chromosome in the population which should be mutated:

X	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Val	**0.96**	**0.78**	**0.46**	**0.45**	**0.13**	**0.15**	**0.63**	**0.58**	**0.93**	**0.48**	**0.02**	**0.98**	**0.85**	**0.80**	**0.33**	**0.61**	**0.27**	**0.34**	**0.42**	**0.31**	**0.35**	**0.82**
r_m^E	0.87	0.94	0.95	0.96	0.90	0.42	0.12	0.88	0.04	0.65	1.00	0.08	0.40	0.62	0.88	0.83	0.86	0.43	0.27	0.61	0.66	0.12

Each gene in the chromosome will be mutated with a probability of mutation $p_m = 0.1$ (mutated genes are highlighted). Gene value after mutation is calculated using Formula (5). Note that each gene has defined its own value of r_m^E. This means that each gene will be mutated with a different random value. Also note that it is not only the optimized variable which mutates, but that control parameter r_m^E is also mutated at the same time. After mutation we get the following chromosome:

X'	***A***										***B***											
Var	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}
Val	**0.96**	**0.78**	**0.46**	**0.45**	**0.00**	**0.15**	**0.63**	**0.58**	**0.93**	**0.48**	**0.02**	**0.98**	**0.85**	**0.80**	**0.43**	**0.61**	**0.27**	**0.34**	**0.42**	**0.31**	**0.35**	**0.80**
r_m^E	0.87	0.94	0.95	0.96	0.55	0.42	0.12	0.88	0.04	0.65	1.00	0.08	0.40	0.62	0.98	0.83	0.86	0.43	0.27	0.61	0.66	0.10

5.4. **Solving real timetable.** Finally, we tested the proposed model on the real timetabling problem at Silesian University in Opava – School of Business Administration in Karvina. The problem size and its structure can be characterized by the values of parameters: number of rooms $n_R = 43$, number of events $n_L = 705$, number of students $n_S = 1807$, number of teachers $n_T = 112$, number of time slots $n_M = 60$, and number of time-room slots $n_G = 2580$.

We ran the genetic algorithm for that model and it was terminated when there was no increase in fitness fiction for 1000 generations, which took about 5 hours on a PC with a 2 GHz processor. The final timetable satisfied all of the hard constraints, i.e., there were no time conflicts for the teachers and all of the events were placed in acceptable rooms.

6. **Conclusions.** In this work a genetic algorithm is presented for CTP. The technique uses an indirect representation featuring some event allocation priorities and invokes a timetable builder routine for constructing a complete timetable. The algorithm incorporates a number of techniques and domain specific heuristic local search operators in order to enhance search efficiency. The non-rigid soft constraints involved in the problem are basically optimization objectives for the search algorithm. A self-adapting principle is also incorporated in the algorithm in order to improve the performance of the GA by controlling the parameters during the evolution. There is an inherent degree of uncertainty involved in objectives which are comprised of different aspects of real life data. This uncertainty is tackled by formulating the measure of violation of soft constraints in the fitness function and using membership functions of trapezoidal fuzzy sets. A special version of the genetic algorithm has been applied on a real world CTP for which a manual feasible solution is already available. It has been shown through an extensive simulation that by incorporating certain combinatorial and domain specific operators the search efficiency of the evolutionary algorithm is significantly enhanced. By comparing the genetic algorithm with the manual solution it is evident that the technique satisfies all of the hard constraints of the problem and achieves a significantly better score in satisfying the soft constraints, and therefore its performance is superior. However, the algorithm is computationally complex when compared to other GA based benchmark heuristics. Furthermore, in order to verify the efficiency and robustness of the algorithm, it should be

tested on different real world timetable problems. The algorithm can also be adapted to solve other CTP and scheduling problems.

7. **Future Work.** In the process of developing CTP through genetic algorithms, all hard and soft constraints are satisfied and significant results are obtained. However, the computational time required is appreciably large. The future work entails the development of well-known heuristics and neuro-fuzzy genetic or rough fuzzy genetic techniques which can reduce the underlying computational complexity so that the quality of the solutions is significantly enhanced.

Another area for future research is a parallel representation of the genetic algorithm. Genetic algorithms are naturally parallel so the implementation can be straightforward. A parallel structure can be used in which there is one master genetic algorithm and N_a slave genetic algorithms. This master genetic algorithm is responsible for performing genetic operators and creating a new generation, while the slave genetic algorithms just calculate the value of the fitness function presented by the master genetic algorithm. In each generation, the master genetic algorithm distributes N_p/N_a tasks (individuals) to each slave genetic algorithm. The master genetic algorithm communicates with the slave genetic algorithms by TCP/IP protocol. Using parallel representation for the proposed genetic algorithm allows us to solve large scale timetabling problems, especially when using a supercomputer which will be built within the framework of the IT4Innovations Centre of Excellence project.

Acknowledgement. This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

REFERENCES

[1] S. C. Brailsford, C. N. Potts and B. M. Smith, Constraint satisfaction problems: Algorithms and applications, *European Journal of Operational Research*, vol.119, pp.557-581, 1999.

[2] E. K. Bruke and S. Petrovic, Recent research directions in automated timetabling, *European Journal of Operational Research*, vol.140, no.2, pp.266-280, 2002.

[3] E. K. Bruke and J. P. Newall, Solving examination timetabling problems through adaptation of heuristic orderings, *Annals of Operations Research*, vol.129, pp.107-134, 2004.

[4] E. K. Burke, B. McCollum and A. Meisels, A graph based hyper heuristic for educational timetabling problems, *European Journal of Operational Research*, vol.176, no.1, pp.177-192, 2007.

[5] T. B. Cooper and J. H. Kingston, The complexity of timetable construction problems, *Lecture Notes in Computer Science*, vol.1153, pp.283-295, 1996.

[6] A. Chaudhuri and K. De, Fuzzy genetic heuristic for university course timetable problem, *Int. J. Advance Comput. Appl.*, vol.2, no.1, pp.100-123, 2010.

[7] P. Cote, T. Wong and R. Sabouri, Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem, *Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science*, vol.3616, pp.151-168, 2005.

[8] D. De Werra, An introduction to timetabling, *European Journal of Operations Research*, vol.19, pp.151-162, 1985.

[9] S. Deris, S. Omatu and H. Ohta, Timetable planning using the constraint-based reasoning, *Computer and Operations Research*, vol.27, pp.819-840, 2000.

[10] D. Dubois and H. Prade, *Possibility Theory: An Approach to Computerized Processing of Uncertainty*, New York, 1988.

[11] A. A. A. Esmin and G. Lambert-Torres, Application of particle swarm optimization to optimal power systems, *International Journal of Innovative Computing, Information and Control*, vol.8, no.3, pp.1705-1716, 2012.

[12] A. Gunawan, K. M. Ng and K. L. Poh, Solving the teacher assignment-course scheduling problem by a hybrid algorithm, *International Journal of Computer and Information Engineering*, pp.137-142, 2007.

[13] A. Gunawan, K. M. Ng and K. L. Poh, A mathematical programming model for a timetabling problem, *Computers and Operations Research*, vol.39, pp.3074-3088, 2012.
[14] S. A. Kazarlis, A. G. Bakirtzis and V. Petridis, A genetic algorithm solution to the unit commitment problem, *IEEE Transactions on Power Systems*, vol.11, no.1, pp.83-92, 1996.
[15] S. A. Kazarlis, S. Papadakis, J. Theocharis and V. Petridis, Micro genetic algorithms as generalized hill climbing operators for genetic algorithm optimization, *IEEE Transactions on Evolutionary Computation*, vol.5, no.3, pp.204-217, 2001.
[16] K. Kim, M. Gen and M. Kim, Adaptive genetic algorithms for multi-resource constrained project scheduling problem with multiple modes, *International Journal of Innovative Computing, Information and Control*, vol.2, no.1, pp.41-49, 2006.
[17] G. Klir and T. Folger, *Fuzzy Sets, Uncertainty and Information*, Prentice Hall, New Jersey, 1988.
[18] A. Konar, *Computational Intelligence Principles, Techniques and Applications*, Springer Verlag, Netherlands, 2005.
[19] D. Kordalewski, C. Liu and K. Salvesen, Solving an exam scheduling problem using a genetic algorithm, *TR-2009-1*, Department of Statistics, University of Toronto, Toronto, Canada, 2009.
[20] R. Lewis and B. Paechter, Application of the grouping genetic algorithm to university course timetabling, *Evolutionary Computation in Combinatorial Optimization, LNCS*, vol.3448, pp.144-153, 2005.
[21] S.-Y. Lin and C.-Y. Chang, Genetic algorithm based iterative two-level algorithm for resource allocation problems and applications, *International Journal of Innovative Computing, Information and Control*, vol.8, no.10(B), pp.7157-7168, 2012.
[22] M. R. Malim, A. T. Khader and A Mustafa, Artificial immune algorithms for university timetabling, *Proc. of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Czech Republic, 2006.
[23] C. Marco, B. Mauro and S. Krzysztof, An effective hybrid algorithm for university course timetabling, *Journal of Scheduling*, vol.9, no.5, pp.403-432, 2006.
[24] R. Perzina, Solving the university timetabling problem with optimized enrollment of students by a parallel self-adaptive genetic algorithm, *Proc. of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, Masarykova Univerzita, pp.264-280, 2006.
[25] R. Perzina, Solving multicriteria university timetabling problem by a self-adaptive genetic algorithm with minimal perturbation, *Proc. of the 2007 IEEE International Conference on Information Reuse and Integration*, pp.98-103, 2007.
[26] J. Ramik and M. Vlach, Generalized concavity in optimization and decision making, *Fuzzy Sequencing and Scheduling*, pp.253-283, 2001.
[27] J. Ramik, A decision system using ANP and fuzzy inputs, *International Journal of Innovative Computing, Information and Control*, vol.3, no.4, pp.825-837, 2007.
[28] A. Scholl and C. Becker, State-of-the-art exact and heuristic solution procedures for simple assembly line balancing, *European Journal of Operation Research*, vol.168, pp.666-693, 2006.
[29] E. Singh, V. D. Joshi and N. Gupta, Optimizing highly constrained examination timetable problems, *Journal of Applied Mathematics, Statistics and Informatics*, vol.4, no.2, pp.193-197, 2008.
[30] S. O. Tasan and S. Tunali, A review of the current applications of genetic algorithms in assembly line balancing, *Journal of Intelligent Manufacturing*, vol.19, pp.49-69, 2008.
[31] M. Tuga, R. Berretta and A. Mendes, A hybrid simulated annealing with Kempe chain neighborhood for the university timetabling problem, *Computer and Information Science*, 2007.
[32] G. White, B. Xie and S. Zonjic, Using tabu search with longer term memory and relaxation to create examination timetables, *European Journal of Operational Research*, vol.153, no.16, pp.80-91, 2004.
[33] L. A. Zadeh, Fuzzy sets, *Information and Control*, vol.8, pp.338-353, 1965.