Solving the University Timetabling Problem with Optimized Enrolment of Students by a Parallel Self-adaptive Genetic Algorithm

Radomír Perzina

Department of Mathematical Methods in Economics,
School of Business Administration, Silesian University,
University Sq. 1934/3, 733 40 Karviná, Czech Republic
perzina@opf.slu.cz

Abstract. The timetabling problem is well known to be NP complete combinatorial problem. The problem becomes even more complex when addressed to individual timetables of students. The core of dealing with the problem in this application is a timetable builder based on mixed direct-indirect encoding evolved by a genetic algorithm with a self-adaptation paradigm, where the parameters of the genetic algorithm are optimized during the same evolution cycle as the problem itself. The aim of this paper is to present an encoding for self-adaptation of genetic algorithms that is suitable for timetabling problem. Comparing to previous approaches we designed the encoding for self-adaptation not only one parameter or several ones but for all possible parameters of genetic algorithms at the same time. Genetic algorithms are naturally parallel so also the parallel representation of the self-adaptive genetic algorithm is presented. The proposed parallel self-adaptive genetic algorithm is then applied for solving the real university timetabling problem and compared with a standard genetic algorithm. The main advantage of this approach is, that it makes possible to solve wide range of timetabling and scheduling problems without setting parameters for each kind of problem in advance. Unlike common timetabling problems the algorithm was applied to the problem in which each student has an individual timetable, so also we present and discuss the algorithm for optimized enrolment of students that minimize the number of clashing constraints for students.

1 Introduction

Genetic algorithms are search algorithms based on the idea of natural selection and natural genetics. It is well known, that efficiency of genetic algorithms strongly depends on their parameters. These parameters are usually set up according to vaguely formulated recommendations of experts or by the so-called two-level genetic algorithm, where at the first-level genetic algorithm optimizes parameters of the second-level. A self-adaptation seems to be a promising way of genetic algorithms, where the parameters of the genetic algorithm are optimized during the same evolution cycle as the problem itself. The aim of this paper is to present an encoding and genetic operators for self-adaptation of genetic algorithms that is suitable for solving the university timetabling problem. Comparing to previous approaches (e.g. [2], [12], [20]) we designed the encoding for self-adaptation not only one parameter but for all or nearly all possible parameters of genetic algorithms at the same time. Moreover, the parameters are encoded separately for each element of a chromosome. Genetic algorithms are naturally parallel so also the parallel representation of the self-adaptive genetic algorithm is presented.

The proposed parallel self-adaptive genetic algorithm is then applied for solving the real university timetabling problem at Silesian University. The problem is known to be NP-complete and hence it is known no algorithm for solving it in polynomial time [8]. The requirements for timetabling differs from university to university, but in general the timetabling problem consists of assigning each lecture from a set of lectures to a suitable room and a time slot subject to a number of hard and soft constraints, such as no teacher can teach more lectures at the same time, at no room can be taught more than one lecture at the same time, teachers time and room preferences, etc.

At some universities including Silesian University each student has an individual timetable, i.e. there are no groups of students, which have the same timetable, even it is difficult to find only two students with the same timetable, thus solving the problem becomes very complex. In order to be able to deal with individual timetables of students we designed an algorithm for optimization of enrollment of students that effectively decrease the number of constraints for student clashes.

A large number of diverse methods have been already proposed in the literature for solving timetabling problems. These methods come from a number of scientific disciplines like Operations Research, Artificial Intelligence, and Computational Intelligence and can be divided into four categories: 1) Sequential Methods that treat timetabling problems as graph problems. Generally, they order the events using domain-specific heuristics and then assign the events sequentially into valid time-room slots [19]. 2) Cluster Methods, in which the problem is divided into a number of event sets. Each set is defined so that it satisfies all hard constraints. Then, the sets are assigned to real time-room slots to satisfy the soft constraints, too [23]. 3) Constraint Based Methods, according to which a timetabling problem is modeled as a set of variables (events) to which values (resources such as teachers and rooms) have to be assigned in order to satisfy a number of constraints [5, 11]. 4) Meta-heuristic methods, such as genetic algorithms, simulated annealing, tabu search, and other heuristic approaches, that are mostly inspired from nature, and apply nature-like processes to solutions or populations of solutions, in order to evolve them towards optimality [1, 7, 14, 18, 21, 22].

When applying genetic algorithms to some optimization or scheduling problem, the crucial element is encoding. For timetabling problem there are two main types of encoding: direct [18] and indirect [14]. The advantage of direct encoding is that the whole search space can be encoded, but it usually leads to violation of many hard constraints. The indirect encoding is based on some rules or instructions for building the resulting timetable and so there is less probability of hard constraint violation, but it can reach only limited portion of the search space and thus it can be trapped in a local optimum. In our application some combination of both encoding was used, because we want to let the algorithm decide itself whether to use the direct or indirect representation, respectively the ratio of using both of them. The timetable builder is based on the order of lectures and time-room slots encoded in the chromosome. By this approach all the feasible timetables can be addressed and the probability of generating an infeasible timetable is strongly reduced. Finally the proposed parallel self-adaptive genetic algorithm is compared to standard genetic algorithms on this timetabling problem. Also the role of enrollment optimization algorithm is discussed.

2 Encoding

Encoding is a crucial element of every genetic algorithm. The structure of our self-adaptive genetic algorithm's encoding is depicted in Figure 1. The idea behind the proposed encoding consists in redundancy of information through hierarchical evaluation of individuals.

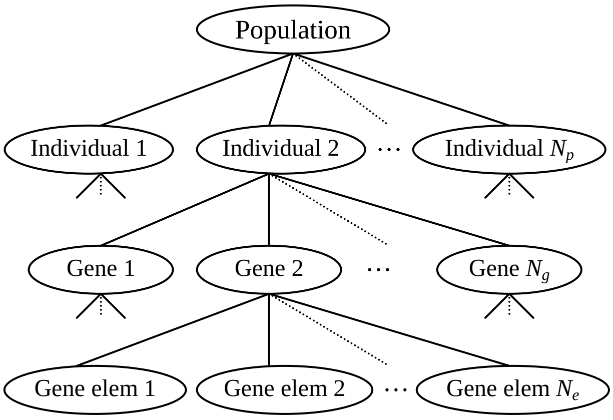

Fig. 1. The structure of a population

As we can see, in the population each individual is composed of N_g genes, where each gene corresponds to exactly one optimized variable. Each gene is composed of N_e gene elements. The number of gene element is different for each gene and it varies through evolution. Each gene element contains low-level parameters, which encode optimized variables and parameters of evolution. All parameters are listed in Table 1.

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 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 altered directly through evolution process, but only indirectly by evaluation mechanism from low level values.

Table 1. The structure of a gene element

Name	Description	Range
x^E	Optimized variable	<0;1>
q_m^E	Parameter of mutation	<-1;1>
q_p^E	Parameter of protected 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>
q_t^E	Parameter of translocation	<-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 2nd partner selection	<0;1>
c_d^E	Coefficient of death	<0;1>
N_p^E	Wanted size of population	<0;1>
i_t^E	Identifier of translocation	<0;1>

Mechanism of Gene Evaluation

The proposed encoding is polyploiditial, so each gene is composed of N_e gene elements. The number of gene elements is variable and undergoes evolution. For evaluation of gene values of gene elements we use simple arithmetical average, i.e.

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

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.

Mechanism of Individual 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 as simple arithmetical average, i.e.

$$X^I = \frac{1}{N_g} \sum_{i=1}^{N_g} X_i^G \tag{2}$$

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

Mechanism of Population Evaluation

Parameters concerning the whole population, such as 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

$$w_f = \frac{N_p^P - i + 1}{\frac{(1 + N_p^P) N_p^P}{2}} \tag{3}$$

where i is index of ith 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, the individual with the second highest value of the fitness function has the value of i equal to 2 etc.

3 Genetic Operators

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

Selection

In genetic algorithms the selection of both parents for mating is usually based on their fitness, but this is not true in nature. In nature a winner of a tournament selects his partner according to his individual preferences. Important is that he cannot take into account his genotype, i.e. directly the values of his genes nor his fitness, but only his 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's preferences for mating and the parameter s_m^I represents individual's phenotype for mating. So the first parent is selected by a tournament selection method with variable ratio of population r_t^P from which the fittest individual is selected. The second parent is selected according to individual's preferences represented by the parameter s_w^I, i.e. the first parent selects an individual with the minimal value of expression $|s_w^I - s_m^I|$, but this selection is made from only limited ratio of population r_p^I.

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

$$X_3^E = X_1^E + \left(X_2^E - X_1^E\right) \cdot r_c^E \tag{4}$$

where X stands for all parameters of a gene element (see Table 1), 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 "$_2$" 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.

Mutation

The mutation operator is applied to every gene element with probability $p_m^E = \left|q_m^E\right|$. Notice that probability of mutation is calculated as the absolute value of the parameter of mutation $q_m^E \in \langle -1;1 \rangle$, 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

$$X_{new}^E = X_{old}^E + \left(X_{\max}^E - X_{\min}^E\right) \cdot U\left(-r_m^E, r_m^E\right) \tag{5}$$

where X stands for all parameters of the gene element, $U(a,b)$ is a random variable with uniform probability distribution in the interval $\langle a;b \rangle$, X_{new}^E is the value of the parameter after mutation, X_{old}^E is the original value of the parameter, $X_{\max}^E$ ($X_{\min}^E$) is the maximal (minimal) allowed bit element value of the parameter as defined in Table 1.

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, that the values of parameter q_u^E of both gene elements are divided by 2, in order to inhibit exponential growth of the number of bit elements.

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 simply removed from the particular gene. By deletion and duplication operators the degree of polyploidity is controlled.

Translocation

The translocation means that a gene element is moved from its original gene to one of neighboring genes with probability $p_t^E = \left|q_t^E\right|$. However, the neighboring gene may decide, whether to accept the gene element, that is the real probability of translocation is defined as

$$p = p_t^E \cdot \left(1 - \left|i_t^{G(new)} - i_t^{G(old)}\right|\right) \qquad \textbf{(6)}$$

where $i_t^{G(new)}$ is the identifier of translocation of the gene, to which the gene element is going to move, and index "$^{(\mathrm{old})}$" denotes the original gene. The values of gene element's parameters are left unchanged with the only exception that q_t^E is multiplied by coefficient -0.5, in order to decrease further translocations. The gene element decides whether to translocate to the left or right neighboring gene according to the sign of q_t^E.

Protected Mutation

Protected mutation is an analogy of local optimization and it is applied only to the fittest individual in the population after application of all previous operators and after values of fitness function of all individuals in the population have been calculated. The protected mutation operator is applied to every gene element with probability $p_p^E = \left|q_p^E\right|$ and after that the new value of fitness function is calculated and compared to the value of fitness function before applying the protected mutation operator. If the new value of fitness function is greater than previous than the mutated chromosome is used otherwise the old chromosome is used for following evolution cycle.

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

$$c_L = 1 - c_d^P\left(1 - w_f\right) \qquad \textbf{(7)}$$

Evidently, through evolution, a less fitter individual causes the 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 killing individuals is repeated until only N_p^P individuals survive in the population.

4 The Parallel Self-adaptive Genetic Algorithm

Genetic algorithms are naturally parallel so it invokes an idea to implement the proposed self-adaptive genetic algorithm parallely. We used simple parallel structure, in which there is one master genetic algorithm and N_a slave genetic algorithms. The master genetic algorithm is responsible for performing genetic operators and creating new generation, while the slave genetic algorithm just calculate the value of fitness function of the individual presented by the master genetic algorithm. In each generation the master genetic algorithm distributes N_p/N_a tasks to each slave genetic algorithm. The master genetic algorithm communicates with slave genetic algorithms by TCP/IP protocol. The structure of the parallel self-adaptive genetic algorithm is depicted in Figure 2.

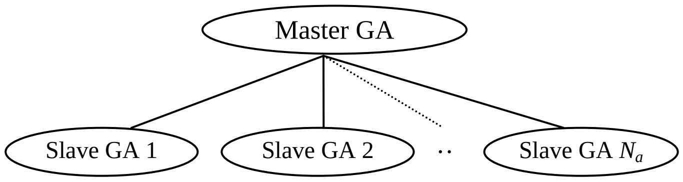

Fig. 2. The structure of the parallel self-adaptive genetic algorithm

5 The Timetabling Problem

In this chapter we describe the input data for the university timetabling problem and formalize the optimization model. In the model we use the following notation

n_R	–	number of available rooms $R_1, R_2, \ldots, R_{n_R}$
n_U	–	number of subjects $U_1, U_2, \ldots, U_{n_U}$
n_L	–	number of lectures (events) $L_1, L_2, \ldots, L_{n_L}$
n_S	–	number of students $S_1, S_2, \ldots S_{n_S}$
n_T	–	number of teachers $T_1, T_2, \ldots, T_{n_T}$
n_M	–	number of time slots $M_1, M_2, \ldots, M_{n_M}$
n_G	–	number of time-room slots $G_1, G_2, \ldots, G_{n_G}$
C	–	clash matrix with elements c_{ij}; $i = 1, 2, \ldots, n_L$; $j = 1, 2, \ldots, n_L$
P	–	preference matrix with elements p_{ij}; $i = 1, 2, \ldots, n_L$; $j = 1, 2, \ldots, n_G$

The purpose of the clash matrix C is to determine which lectures should not be scheduled at the same time. Each element of the clash matrix c_{ij} is equal to the number of students, which are enrolled to both lectures L_i and L_j. The number of students that attend more lectures at the same time is only soft constraint, because each student has an individual timetable and so it is nearly impossible to build a timetable with no clashes for students. The clash matrix C is also used for handling teachers clashes, i.e. the high penalty coefficient is set to the matrix element $c_{ij} = 10^6$ for all lectures L_i and L_j which are taught by the same teacher.

The purpose of the preference matrix P is to set preference p_{ij} to particular time-room slot G_j for each lecture L_i. Note that the preferences are taken as negative preferences, i.e. the higher is the element of the preference matrix p_{ij}, the less suitable is the time-room slot G_j for the lecture L_i. The matrix P is also used for handling the suitable rooms for each lecture. By the matrix P we can handle both, the teacher time preferences and the suitable rooms for each lecture. The teacher time preferences are usually taken as soft constraints, i.e. only small penalty coefficient is set to p_{ij} for all time-room slots G_j that correspond to less preferred timeslots for the lectures L_i that are taught by the teacher. The requirements for suitable room is handled as the hard constraint, so the high penalty coefficient $p_{ij} = 10^6$ is set for all time-room slots G_j that correspond to unsuitable rooms for particular lecture L_i.

The core of the timetabling problem is to assign suitable timeslot G_j to each lecture L_i such that all hard constraints were satisfied and the number of soft constraint violations was minimal. This problem can be mathematically formulated as optimization model minimizing error of the timetable defined as

$$z = \sum_{i=1}^{n_L} \left[\sum_{j=1}^{n_G} x_{ij} p_{ij} + \sum_{k=1}^{n_L} c_{ik} \, \mathrm{sametime}(i,k) \right] + w_p \cdot \mathrm{penalty}(x_{ij}) \rightarrow \min \tag{8}$$

s.t. $$\sum_{i=1}^{n_L} x_{ij} \le 1 \quad \text{for } j = 1, 2, \ldots, n_G$$

$$\sum_{j=1}^{n_G} x_{ij} = 1 \quad \text{for } i = 1, 2, \ldots, n_L,$$

where x_{ij} is a binary optimized variable determining whether the lecture L_i is taught in the time-room slot G_j. The expression sametime(i,k) is the function that is equal to 1 if the lecture L_i is taught at the same time as the lecture L_k, otherwise it is equal to zero. The expression penalty(x_{ij}) is the function determining penalty of the timetable that is not possible to express by clash matrix C or preference matrix P. And the coefficient w_p is the weight of penalty(x_{ij}) by which it contributes to the error of the timetable.

6 Teacher Preferences

To express teacher preferences we define following criteria. For each criterion the teacher sets its preference t^* by “mark” from 1 to 5, the mark 1 means that it is the best and mark 5 means that it is the worst. Because the preferences t^* are used for calculation of total error of the timetable and if $t^* = 1$ it means the best possibility for the teacher and actually no error of timetable, we transform t^* to $t^{*'}$ by decreasing 1, i.e. $t^{*'} = t^* - 1$. If the preference $t^* = 5$ it is hard constraint, so $t^{*'} = 10^6$.

Time Preferences

For each timeslot M_k the teacher must set its preference t_k^M, for which we calculate $t_k^{M'} = t_k^M - 1$. After that we assign $p_{ij} = w_M \cdot t_k^{M'}$ for all lectures L_i that are taught by this teacher and for all time-room slots G_j that corresponds to timeslot M_k. The parameter w_M is a weight by which it contributes to the error of the timetable. For example if the teacher cannot teach on Wednesdays, on Mondays and Tuesdays he prefers to teach in the afternoon and on Thursdays and Fridays he prefers to teach in the morning, then the preferences t_k^M could look like that:

	08:00-08:45	08:50-09:35	09:40-10:25	10:30-11:15	11:20-12:05	12:10-12:55	13:00-13:45	13:50-14:35	14:40-15:25	15:30-16:15
Mon	*4*	*3*	*3*	*3*	*2*	*2*	*1*	*1*	*1*	*1*
Tue	*4*	*3*	*3*	*3*	*2*	*2*	*1*	*1*	*1*	*1*
Wed	*5*	*5*	*5*	*5*	*5*	*5*	*5*	*5*	*5*	*5*
Thu	*1*	*1*	*1*	*1*	*1*	*2*	*2*	*3*	*3*	*3*
Fri	*1*	*1*	*1*	*1*	*1*	*2*	*2*	*3*	*3*	*3*

Number of Teaching Days per Week

For each number of days the teacher must set t_k^N, for which we calculate $t_k^{N'} = t_k^N - 1$. We calculate the number of days *d* in which the teacher teaches at least one lecture and then increase the value of penalty(x_{ij}) by the value of $w_N \cdot t_d^{N'}$, where w_N is the weight by which it contributes to the penalty of timetable. For example if the teacher would like to teach in 2 or 3 days per week, in 4 days it is not convenient for him, in 5 days it is not acceptable for him and in 1 day it is not possible to teach all lectures then the preferences t_k^N could look like that:

Number of Teaching Days	1	2	3	4	5
Preferences t_k^N	*5*	*1*	*1*	*3*	*5*

Length of Teaching Block without Break

By this criterion the teacher sets if he prefers to concentrate lectures to one long teaching block or to disperse it to several short teaching blocks. For each length of the teaching block (in hours) the teacher must set its preference t_k^B, for which we calculate $t_k^{B'} = t_k^B - 1$. We calculate for each continuous teaching block its length *l* and then increase the value of penalty(x_{ij}) by the value of $w_B \cdot t_l^{B'}$, where w_B is the weight by which it contributes to the penalty of timetable. For example if the teacher would not like to have too dispersed lectures, i.e. he wants to teach at least 2 hours without break, the most preferably he would like to teach 3-4 continuous hours, 5 continuous hours is very exhausting and more than 5 continuous hours is not possible to teach then the preferences t_k^B could look like that:

The Length of Block	1	2	3	4	5	6	7	8	9	10
Preferences t_k^B	*5*	*3*	*1*	*1*	*3*	*5*	*5*	*5*	*5*	*5*

Number of Teaching Hours per Day

For each number of the teaching hours the teacher must set its preference t_k^H, for which we calculate $t_k^{H'} = t_k^H - 1$. We calculate for each day the number of teaching hours *h* and then increase the value of penalty(x_{ij}) by the value of $w_H \cdot t_h^{H'}$, where w_H is the weight by which it contributes to the penalty of timetable. For example for the teacher it is very inconvenient to go to school to teach only 1 or 2 hours, optimal number is 3-5 hours per day, 6-7 hours is exhausting and above 6 hours per day is impossible then the preferences t_k^H could look like that:

The Number of Hours	1	2	3	4	5	6	7	8	9	10
Preferences t_k^H	*5*	*4*	*2*	*1*	*1*	*2*	*3*	*5*	*5*	*5*

Span of Teaching Day

This criterion means the difference between beginning of the first lecture and the end of last lecture in a day, i.e. the sum of teaching hours and breaks between them. For each length of span the teacher must set its preference t_k^S, for which we calculate $t_k^{S'} = t_k^S - 1$. We calculate for each teaching day the length of span *s* and then increase the value of penalty(x_{ij}) by the value of $w_S \cdot t_s^{S'}$, where w_S is the weight by which it contributes to the penalty of timetable.

Length of Continuous Break

By this criterion the teacher sets how many hours he needs to relax. For each number of relax hours the teacher must set its preference t_k^R, for which we calculate $t_k^{R'} = t_k^R - 1$. We calculate for each break between two teaching blocks its length *r* and then increase the value of penalty(x_{ij}) by the value of $w_R \cdot t_r^{R'}$, where w_R is the weight by which it contributes to the penalty of timetable. Of course if necessary it is possible to incorporate other teacher preferences in similar way as previous ones.

7 Enrollment of Students

At most universities there are some group of students which share the same timetable. But at some universities including Silesian University each student has an individual timetable, i.e. there are no groups of students, which have the same timetable, even it is hardly to find only two students that have the same timetable, thus solving the problem becomes very complex.

At Silesian University each student can choose subjects that he wants to study. If the subject consists of only one lecture there is usually no problem as the student is automatically enrolled to that lecture. But most of subjects consist of two kinds of lectures: classical lectures and seminaries. There are usually more seminaries of the same subject, but the student can be enrolled only to one of them. The question is how to set appropriate seminary for each student. One possibility is to do it randomly, but by this way it will be very difficult or nearly impossible to build the timetable, in which each student has unclashing timetable or the number of clashing lectures for students is acceptably small. So we propose the algorithm for optimized enrolment of students that minimize the number of clashing constraints for students.

In the model we use the following notation

S_{ij}^{U}	–	binary variable defining whether the student S_i is enrolled to the subject U_j
S_{ij}^{L}	–	binary variable defining whether the student S_i is enrolled to the lecture L_j
U_{ij}^{L}	–	binary variable defining whether the subject U_i contains the lecture L_j
S_{ij}^{L}	–	binary variable defining whether the student S_i is enrolled to the lecture L_j
L_i^S	–	maximal number of students that can be enrolled to the lecture L_i

Without loss of generality suppose that each kind of lecture of the same subject will be labeled as different subject. First we set the elements c_{ij}^T of clash matrix C for teacher clashes as was described in the chapter 5. After that student are enrolled to the lectures corresponding to the subjects which have only one lecture of the same type, i.e. there is no possibility of choice of the lecture to which the student should be enrolled. The core of the enrollment problem is then to enroll all students to all lectures such that all constraints were satisfied and the number of nonzero elements c_{ij} of the clash matrix C was minimal. The problem can be mathematically formulated as optimization model minimizing the number of nonzero elements c_{ij} defined as

$$c = \sum_{i=1}^{n_L}\sum_{j=1}^{n_L} \mathrm{nonzero}(c_{ij}) \to \min \tag{9}$$

s.t. $$\sum_{i=1}^{n_S} S_{ij}^L \le L_j^S \quad \text{for } j = 1, 2, \ldots, n_L$$

$$\sum_{j=1}^{n_U} S_{ij}^U = \sum_{k=1}^{n_L} S_{ik}^L \quad \text{for } i = 1, 2, \ldots, n_S$$

$$\sum_{j=1}^{n_L} S_{ij}^L \cdot U_{kj}^L \cdot S_{ik}^U = 1 \quad \text{for } i = 1, 2, \ldots, n_S,\ k = 1, 2, \ldots, n_U$$

where $c_{ij} = c_{ij}^T + \sum_{k=1}^{n_S} S_{ki}^L \cdot S_{kj}^L$ for $i, j = 1, 2, \ldots, n_L$ and nonzero(c_{ij}) is the function which is equal to 0 when c_{ij} is zero and 1 otherwise.

8 Mapping the Timetabling Problem to the Chromosome

The timetabling problem actually consists of two tasks. In the first one students must be enrolled to lectures and in the second one there must be lectures assigned to time-room slots. In this section the process of decoding from the chromosome will be described.

The Enrolment Builder

First we enroll students to lectures according their preferences for subjects they want to attend by the process described in the chapter 7. For optimization of enrollment of students the proposed parallel self-adaptive genetic algorithm was used. As was mentioned in the chapter 2, each gene of a chromosome represents one real variable within the interval <0;1>. In order to apply this chromosome encoding for the enrollment problem, the chromosome is divided into two parts. The first part *A* consisting of $(n_S \cdot n_U)$ genes represents the parameters for all subjects selected by students and the second part *B* consisting of n_L genes represents parameters for lectures. The main idea behind the encoding of the lecture enrollment is that the subjects selected by students are sorted in ascending order according to values of parameters in the part *A* of the chromosome and then in this order the lecture enrollment builder assigns the first free suitable lecture with the least difference of $\left|A_i - B_j\right|$, where A_i is the *i*-th parameter of the part *A* of the chromosome and B_j is the *j*-th parameter of the part *B* of the chromosome. The fitness function *f* for the genetic algorithm is the negative value of *c* in (9), i.e. $c = -z$.

The Timetable Builder

For solving the university timetabling problem the parallel self-adaptive genetic algorithm was used, too. The process of encoding is similar to encoding of the enrollment problem above. The chromosome is divided into three parts. The first part *A* consisting of n_L genes represents the parameters for lectures, the second part *B* consisting of n_G genes represents parameters for time-room slots and the last part contains control parameters for the timetable builder. Lectures are sorted in ascending order according to values of parameters in the part *A* of the chromosome and then in this order the timetable builder assigns the first suitable unused time-room slot with the least difference of $\left|A_i - B_j\right|$, where A_i is the *i*-th parameter of the part *A* of the chromosome and B_j is the *j*-th parameter of the part *B* of the chromosome. Whether the time-room slot G_j is suitable for the lecture L_i is determined by the control parameter *D* in the last part of the chromosome. The parameter *D* contains the maximal accepted penalty of assigning lecture L_i to time-room slot G_j, which is calculated by the formula (8). If there is no suitable time-room slot for the lecture L_i, the best suited still unused time-room slot is selected for the lecture. The fitness function *f* for the genetic algorithm is the negative value of *z* in (8), i.e. $f = -z$.

To make the idea behind decoding the chromosome more clear, a simple example will be provided. Let's suppose we have three lectures: L_1, L_2, L_3 and four time-room slots: G_1, G_2, G_3, G_4. So the chromosome for such simple timetable will have 8 genes. Let's suppose that after evaluation, the gene values of parameters x^E are:

Part	*A*			*B*				*D*
Description	L_1	L_2	L_3	G_1	G_2	G_3	G_4	
Value	0,45	0,91	0,39	0,82	0,36	0,49	0,56	0,8

First lectures must be sorted according values of x^E, so the order will be L_3, L_1, L_2. For all lectures we now must calculate difference between the lecture and particular time-room slot $|A_i - B_j|$:

	G1	***G2***	***G3***	***G4***
L3	0,43	**0,03**	0,10	0,17
L1	0,37	*0,09*	**0,04**	0,11
L2	**0,09**	*0,55*	*0,42*	0,35

The selected time-room slot with least difference of $|A_i - B_j|$ for each lecture is marked by bold font and time-room slots used for previous lectures are in italic. So the resulting timetable according chromosome provided in the example will look like that: L1→G3, L2→G1, L3→G2.

9 Numerical Experiments

This model was then applied for solving the real timetabling problem in the School of Business Administration at Silesian University. The problem size and its structure can be characterized by the values of parameters: number of rooms $n_R = 43$, number of subjects $n_U = 340$, number of lectures $n_L = 705$, number of students $n_S = 1807$, number of teachers $n_T = 112$, number of time slots $n_M = 60$, number of time-room slots $n_G = 2400$. When evaluating the error of timetable z defined in (8), we must set up weights of the criteria: $w_M = 3$, $w_N = 5$, $w_B = 3$, $w_H = 3$, $w_S = 2$, $w_R = 2$, $w_P = 0.5$. The number of computers that were used was $N_a = 30$.

The best solution found by the parallel self-adaptive genetic algorithm was the timetable with the minimal value of error function $z = 7184$. The resulting timetable satisfied all hard constraints and there were 83 students that had any clashing lecture. Previously used approach for constructing the timetable produced the timetable, in which there were in average 2.8 clashing lectures for each student, moreover it was very boring and time consuming process, because the timetable was completely made manually, computer was used only as graphical user interface.

In order to test also the performance of the proposed self-adaptive genetic algorithm (SAGA) we have compared it with the simple genetic algorithm (SGA) on this timetabling problem. The simple genetic algorithm used a binary encoding, the size of population was 30 individuals, probability of mutation 0.003 and elitism was used.

Maximal number of generations for both algorithms was 10^4. We ran both algorithms 10 times and measured the average penalty function *z* of the best timetable found in each ran of both genetic algorithms. The average best value of error function for SAGA was 7331 and for SGA the average value of *z* was 7687. As we can see SAGA was slightly better, but the main advantage of SAGA is that there is no need for finding values of the parameters, as there are no parameters set in advance.

We also tested the role of enrollment optimization algorithm. The enrolment optimization algorithm as described in the chapter 7 was substituted by random enrollment students to lectures and the best solution found by the parallel SAGA was the timetable with the minimal value of error function $z = 12553$. As we can see it is much worse than with applying the enrollment optimization algorithm.

10 Conclusions

In this paper we have designed the optimization model for solving the university timetabling problem that is capable of dealing with individual timetables of every student. For solving the timetabling problem we have proposed a parallel self-adaptive genetic algorithm with self-adaptation of all its parameters. This algorithm was applied for solving the real university timetabling problem at Silesian University. It was shown that the parallel self-adaptive genetic algorithm is able to effectively solve the timetabling problem. It was also shown how to significantly decrease the number of student clash constraints by the proposed enrollment optimization algorithm when dealing with individual timetables of students.

Great problem has appeared when it was applied to the real timetabling problem with changed preferences and requirements for timetable, because the new timetable completely different comparing to the original one. So the further study will be concerned to deal with the problem of minimization of number of changes between new and original timetable.

References

1. Abramson, D. *Constructing school timetables using simulated annealing: sequential and parallel algorithms European Journal of Operational Research.* Management Science, vol. 37, no. 1, January 1991, pp. 98-113.
2. Bäck, T. *Self-Adaptation in Genetic Algorithms*. Proceedings of the First European Conference on Artificial Life. Cambridge: MIT Press, 1992.
3. Bufé, M. et al. *Automated Solution of a Highly Constrained School Timetabling Problem – Preliminary Results.* In Proceedings of the Evo Workshops 2001, Como, Italy: Springer, 2001.
4. Burke, E., Newall, J. Enhancing timetable solutions with local search methods. Lectures Notes in Computer Science 2740, pp. 195-206, Berlin: Springer – Velag, 2003.

5. Brailsford, S. C. et al. *Constraint Satisfaction Problems: Algorithms and Applications*. European Journal of Operational Research, vol 119, 1999, pp. 557-581.
6. De Jong, K. A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD. Thesis, University of Michigan, 1975.
7. Di Gaspero, L., Schaerf, A. *Tabu search techniques for examination timetabling.* Lectures Notes in Computer Science 2079, pp. 104-117, Berlin: Springer – Velag, 2001.
8. Even, S., Iati, A., Shamir, A. *On the Complexity of Timetabling and Multicomodity Flow Problems.* Siam Journal of Computation, vol. 5, no. 4, pp. 691-703. 1976.
9. Fernandes, C. et al. *High School Weekly Timetabling by Evolutionary Algorithms*. In Proceedings of 14th Annual ACM Symposium on Applied Computing. San Antonio, Texas, 1999.
10. Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Massachusetts: Addison Wesley Readings, 1989.
11. Legierski, W. *Search Strategy for Constraint-Based Class-Teacher Timetabling.* Lectures Notes in Computer Science 2740, pp. 247-261, Berlin: Springer – Verlag, 2003.
12. Marsili, S. L., Alba, P. A. *Adaptive Mutation in Genetic Algorithms*. Soft Computing, vol. 4, no. 2, pp. 76-80. New York: Springer – Verlag, 2000.
13. Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd ed. New York: Springer – Verlag, 1996.
14. Paechter, B. et al. *Timetabling the Classes of an Entire University with an Evolutionary Algorithm*. Parallel Problem Solving from Nature (PPSN) V. Lectures Notes in Computer Science 1498, pp. 865-874, Berlin: Springer – Verlag, 1998.
15. Perzina, R. *Self-adaptation in Genetic Algorithms*. In Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003), pp. 234-238. Orlando, FL, USA: IIIS, 2003.
16. Perzina, R. *A Self-adapting Genetic Algorithm for Solving the University Timetabling Problem*. In: Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), pp. 284-288. Orlando, USA: IIIS, 2004.
17. Perzina, R., Ramík, J. *A New Portfolio Selection Model Solved by Genetic Algorithms*. Proceedings of 20th International Conference MME 2002, pp. 201-207. Ostrava: VŠB TU, 2002.
18. Ross, P., Hart, E., Corne, D. *Some Observations about GA-based Exam Timetabling.* In The Practice and Theory of Automated Timetabling: Selected Papers from the Second International Conference. Lecture Notes in Computer Science 1408, pp. 115-129, Berlin: Springer – Verlag, 1997.
19. Schaerf, A. *A Survey of Automated Timetabling.* Artificial Intelligence Review, vol. 13, no. 2, pp. 87-127, Norwell: Kluwer Academic Publishers, 1999.
20. Stefano, C. D., Tettamanazi, A. G. B. *An Evolutionary Algorithm for Solving the Schoool Timetabling Problem*. In Proceedings of the Evo Workshops 2001, pp. 452-462, Como, Italy: Springer, 2001.
21. Terashima, H. et al. *Evolution of Constraint Satisfaction Strategies in Examination Timetabling.* In Proceedings of the Genetic and Evolutionary Computation Conference GECCO 1999, pp. 635-642, Morgan Kaumann, 1999.

22. Thompson, J.M., Dowsland, K.A. *A robust simulated annealing based examination timetabling system.* In Computers and Operations Research, vol. 25, pp. 637-648, Oxford: Elsevier Science Ltd., 1998.
23. White, G. M., Chan, P. W. *Towards the Construction of Optimal Examination Timetables*. INFOR 17, 1979, p.p. 219-229.
24. Yang, J. M., Kao, C. Y. *Integrating Adaptive Mutations and Family Competition into Genetic Algorithms as Function Optimizer*. Soft Computing, vol. 4, no. 2, pp. 89-102. New York: Springer – Verlag, 2000.