Mathematics and Statistics 8(6): 728-739, 2020 http://www.hrpub.org
DOI: 10.13189/ms.2020.080614

Algorithmic Verification of Constraint Satisfaction Method on Timetable Problem

Viliam Ďuriš

Department of Mathematics, Constantine the Philosopher University in Nitra, Tr. A. Hlinku 1, 94974 Nitra, Slovakia

Received October 14, 2020; Revised November 30, 2020; Accepted December 20, 2020

Cite This Paper in the following Citation Styles

(a): *[1] Viliam Ďuriš , "Algorithmic Verification of Constraint Satisfaction Method on Timetable Problem," Mathematics and Statistics, Vol. 8, No. 6, pp. 728 - 739, 2020. DOI: 10.13189/ms.2020.080614.*

(b): *Viliam Ďuriš (2020). Algorithmic Verification of Constraint Satisfaction Method on Timetable Problem. Mathematics and Statistics, 8(6), 728 - 739. DOI: 10.13189/ms.2020.080614.*

Abstract Various problems in the real world can be viewed as the Constraint Satisfaction Problem (CSP) based on several mathematical principles. This paper is a guideline for complete automation of the Timetable Problem (TTP) formulated as CSP, which we are able to solve algorithmically, and so the advantage is the possibility to solve the problem on a computer. The theory presents fundamental concepts and characteristics of CSP along with an overview of basic algorithms used in terms of its solution and forms the TTP as CSP and delineates the basic properties and requirements to be met in the timetable. The theory in our paper is mostly based on the Jeavons, Cohen, Gyssens, Cooper, and Koubarakis work, on the basis of which we've constructed a computer programme, which verifies the validity and functionality of the Constraint satisfaction method for solving the Timetable Problem. The solution of the TTP, which is characterized by its basic characteristics and requirements, was implemented by a tree-based search algorithm to a program and our main contribution is an algorithmic verification of constraints abilities and reliability when solving a TTP by means of constraints. The created program was also used to verify the time complexity of the algorithmic solution.

Keywords NP-complete Problem, Timetable Problem, Computational Complexity, Decision Problem, Constraint Satisfaction, Constraint Satisfaction Problem, Algorithms for CSP, School Timetabling, Optimization, Backtracking, Relational Algebra

1. Constraint Satisfaction Problem

Thanks to appropriate algorithms, we can solve difficult problems which we call NP -complete [1], whose analytical calculation would be impossible. This class includes a lot of other mathematical problems (*Boolean satisfiability problem*, *Travelling Salesman problem* or *Graph k-coloring problem*, etc.). However, in order to successfully solve the problems that we encounter in everyday practice, we need to be able to characterize each of them appropriately so as to find a mathematical prescription for its solution.

The Timetable Problem is nowadays solved on the computer (rarely manually), but most often only semi-automatically at the level of the timetable creator's activity method. The creator controls a smart editor – a programme that supports and controls his or her activities. The programme is usually designed to offer possible partial solutions (ensuring parallel supervision of timetable creation in the environment of educators, study groups and classrooms) to assist in the selection of possible solutions (which arise from "overlapping" timetables of individual environments), allowing the selected solution to be registered in all three environments (or accept changes in the timetable) or monitor the status of the timetable creation process (how many subjects have not yet been included, how many conflicts have been created during the timetable creation and suchlike).

A number of important problems in mathematics (e.g. in graph theory, set theory, algebra, etc.) and in everyday life can be easily described as so-called *Constraint Satisfaction Problems* (CSP) with the appropriate

constraint language, a set of relations over a set of values. There is a relationship between the "power of language" and the problems that can be expressed in some language [2]. The advantage of formulating the problem as a CSP due to the commonly disparate structure of the problem is the ability to work out any problem as a mathematical object and to solve any problem algorithmically. In this way, a purely theoretical solution to the problem becomes applicable to almost anything. We can understand CSP as trying to find such a solution from the information we have about the problem, so that all the conditions (constraints) of the problem are met. The constraints determine the properties of the solution, and most of the time, they create a whole set of possible solutions. A characteristic feature of the general CSP is that it is an NP -complete problem [3] that cannot be solved analytically. Therefore, it is typical for solutions using various algorithms by taking advantage of computer "power", and this part of mathematics (closely related to computer science) has gained importance with the advent of computers. A typical and known CSP is the Eight Queens Problem [4], [5].

For simplicity, every problem is made up of a set of variables that are displayed in a set of values. Then, all possible assignments of values to variables that are considered in CSPs are called a state space, and the problem can be solved by searching in the state space. The state space of the problem can be imagined as a plot graph of the n-dimensional function projected into the plane by suitable mapping, whose vertices correspond to the state of the problem solution and the vertices' lines (edges) correspond to the operations changing the state to another. The solution to the problem then has the nature of searching for possible paths between the vertex corresponding to the initial state of the problem solution (e.g. randomly selected) and the vertex corresponding to the target state conditions. The best solution is always presented by the vertex that is the global minimum of the function, and therefore the searched function must be thus constructed [6] [7].

When searching in the state space, we can only be interested in finding a solution or finding the best solution (global minimum). Often (e.g. because of the size of the problem) the problem is simply divided into related subproblems (if this is permitted by the problem itself in any way), which we can solve more efficiently, and then we can suitably put them together to solve the original problem (e.g. using the CSP Graph Structure). This greatly improves the search for a solution, but such a problem-division strategy is only effective in most cases. In addition, it always depends on the choice of algorithm that primarily influences finding the solution and the efficiency of the search itself.

The theory of constraints is currently dedicated to how to define constraints, what relationships must apply between them, and how restrictive they may be (sufficient and necessary constraints) to make the complex problem practically solvable. However, mathematical theory is built as algebraic, graph and database one. The main direction in future research is mostly the application of algebra knowledge [8]. The more that is known about valid relationships between problems, the better any problem can be mathematically described, simplified, or transferred into an equivalent problem, and the sooner we will be able to deal with it and completely solve any problem as a practically solvable task. Constraint programming is becoming a key technology for solving many combinatorial problems.

2. CSP Mathematical Model

Let x be a variable and D a set of values that variable x can acquire. Then, D is called *domain* of variable x. The symbol $|D|$ denotes the number of elements of domain D. Let $D_i, i = 1, \dots, n, n \in N$, be domains. Then the set of all ordered n-tuples $[x_1, \dots, x_n], x_i \in D_i, 1, \dots, n$, are called the *Cartesian product* of sets $D_i, 1, \dots, n$. We denote them $D_1 \times D_2 \times \dots \times D_n$. If $\forall i D_i = D$, the set $D \times \dots \times D = \{[x_1, \dots, x_n], x_i \in D, i = 1, \dots, n\}$ is called the *Cartesian power* of set D and is denoted D^n. Let D be a domain and n a natural number. Then, by *n-ary relation* over the set D, we call each subset R of the set D^n.

The constraint is then characterized by a certain relationship between the variable values. Let us consider two variables denoted as x_1 and x_2 and the set of values $D = \{d_1, d_2, d_3\}$. Then, the constraint in terms of variables x_1, x_2 must be assigned mutually different values from set D and can be expressed as this set:

$$\{\{(x_1, d_1), (x_2, d_2)\}, \{(x_1, d_1), (x_2, d_3)\}, \{(x_1, d_2), (x_2, d_1)\}, \{(x_1, d_2), (x_2, d_3)\}, \{(x_1, d_3), (x_2, d_1)\}, \{(x_1, d_3), (x_2, d_2)\}\}$$

However, such an interpretation of the constraint is not advantageous in that it is not immediately clear what characterizes the relationship between the values of the variables and what the particular variables are. It is therefore necessary to separate these two aspects. So let D be a domain and n a natural number. Under *constraint* c, we mean a pair (p, R) where p is an ordered n-tuple of variables and R an n-ary relation over the domain D. N-tuple p is called the *field* of the constraint, and the number of elements (n) in the relation the *arity* of the constraint. Then, with relation R, all allowed combinations are exactly determined with respect to the variables in field p. Thus, the unary constraint specifies all allowed values for one variable, and the binary constraint does so for all allowed combinations of values for the ordered pair of variables. As a result, the above example of the constraint would be written as the set:

$$([x_1, x_2], \{[d_1, d_2], [d_1, d_3], [d_2, d_1], [d_2, d_3], [d_3, d_1][d_3, d_2]\})$$

or

$$([x_1, x_2], \{[c_1, c_2]; c_1, c_2 \in \{d_1, d_2, d_3\} \wedge c_1 \neq c_2\}).$$

The second notation of constraint is particularly useful when defining the constraint over infinite domains (which is not the case with most CSPs we need to solve in practice). *Meeting the constraint* means to assign such an n-tuple of values (or a combination) $[d_1, \dots, d_n]$ to the field of variables that are from the relation. We write $c(d_1, \dots, d_n) = |d_1, \dots, d_n| = 1$. Then, the CSP consists of variables that acquire values from the associated domains and the constraints placed on the variables. If the constraint is met, we say that the values assigned to the variables are *consistent*. Otherwise, they are *inconsistent*.

The constraint fulfillment process is an envisaged solution to the problem as a gradual fulfillment of specified constraint conditions. In the solution procedure, the initial set of values from the domain is gradually kept under constraint until a set satisfying the specified constraints is obtained. If the domain values are not consistent with the given and derivable constraints [9], [10], [11], they are removed from the domain, leaving in it only those that meet all the constraints. Gradually, by meeting the constraints, we get a set corresponding to the target state, or we find out that the problem has no solution. Thus, the consistency of variables is preserved by this process.

We can say that CSP is $P = (X, D, C)$ consisting of:

1. a finite set of variables $X = \{x_1, \dots, x_n\}, n \in \mathbb{N}$
2. a finite set of finite and discrete domains $D = \{D_1, \dots, D_n\}$, where D_i is a domain for variable $x_i, i = 1, \dots, n$
3. a set of constraints $C = \{c_1, \dots, c_m\}, m \in \mathbb{N}$ over the set X

It is not limiting if, instead of the finite set of finite and discrete domains, we consider only one final domain for all variables (if they have the same structure). Indeed, if the variables had different domains, the domain would be the unification of all domains. If the sets X and D are final, the CSP is called the *Finite Constraint Satisfaction Problem* (also referred to as $FCSP$). For each CSP, we try to solve the following subproblem:

1. decide whether there exists a solution
2. find the solution (if any)
3. determine the number of all solutions
4. find all solutions

Each of these problems has a different complexity (and therefore a different choice for the solution search process), so it is always necessary to realize which subproblem we are trying to solve before solving the problem itself. Now, let $X = \{x_1, \dots, x_n\}, n \in N$ be a set of variables. Let $D = \{d_1, \dots, d_k\}, k \in N$ be a finite domain for variables $x_i, i = 1, \dots, n$. Then, mapping $h: X \to D$ of the set of variables into the set of variable values the way that $\forall i = 1, \dots, n: h(x_i) = d_j, d_j \in D, j = 1, \dots, k$ is called the *interpretation of variables* of set X. The *solution* of the particular CSP is such an interpretation of the variables that each constraint is fulfilled (i.e. each field is an element of the relevant constraint relation). The mapping can be written down as $h: X \to D$, že $\forall i, i = 1, \dots, m: h(p_i) \in R_i$ (m is the number of constraints, $h(p_i)$ is the result of applying the function h to the field p_i, if, for example, $p_i = [x_1, \dots, x_k], k \in N, h(p_i) = [h(x_1), h(x_2), \dots, h(x_k)]$). Formally, we write down $\forall i, i = 1, \dots, m: |h(p_i)| = 1$. The set of solutions is denoted $Sol(P)$. Next, let R_D be a set of all finite relations over set D. Then, for each set of relations $\Gamma \subseteq R_D$ the class of all CSPs whose constraint relations are elements of the set Γ is denoted $CSP(\Gamma)$. The set Γ specifying the CSP is called a *constraint language*. Simply, the more relations that are included in the constraint language, the more constraints can be expressed. This property is called the *power of language*. Naturally, if we increase the power of the language, the corresponding problem becomes more difficult to solve.

If there is an algorithm that solves every instance of the problem in class $CSP(\Gamma)$ by polynomial time complexity, class $CSP(\Gamma)$ is a class of P problems, and the set of relations Γ is a so-called set of practically solvable relations. On the other hand, if $CSP(\Gamma)$ is NP-complete, we say that the constraint language Γ is NP-complete [12].

3. Possible Approaches to a CSP Solution

In order to solve a CSP, we need to find such an assignment of values to the variables so that all the required constraints are met for the solution. For small values $|D|$ or in the case of "simple" problems, we can decide about CSP by analytical calculation. However, once the number $|D|$ is increased (for most CSPs it is enough to talk about $|D| > 2$), it becomes "inconvenient" to look for the solution itself or even just the number of solutions for the class of P problems. On the other hand (in the knowledge of deeper CSP theory), it is possible to find at least partial (incomplete) solutions of some CSPs on the basis of various simplistic relationships or the equivalence between problems.

So far, the only possible way for NP-complete problems is using algorithms. In this section, we will describe the *BACKTRACKING* algorithm as a possible approach to a CSP solution. The choice of the correct algorithm is dependent on the specific CSP instance and structure. There is no universal prescription that is able to determine which algorithm to use and which has a greater "power". It may happen that a CSP class is better addressed by an algorithm that would not be "the one prescribed". And even just some instance of the class. It also depends on whether we are looking for only the number of solutions or the solution itself, or even the best solution. Thus, there is an additional problem with the

CSP, which is to decide which algorithm is to be used to solve the CSP. Various problem-solving techniques have been studied for decades, and efforts are currently concentrated mainly in the field of genetic algorithms or various local search methods [13], [14].

We know that the problem is solved by searching the state space, which can be *complete* or *incomplete*. A *complete search* of the state space is such a search that finds a CSP solution or finds that a solution does not exist. An *incomplete search* of the state space search is a search that is not complete. Thus, an incomplete search may or may not find a solution, even if the solution exists (or, it finds out that the problem does not have a solution). Alternatively, it finds just an incomplete (partial) solution to the problem (depending on the nature of the CSP). The big advantage of an incomplete search is its speed. *A partial solution* to a CSP is such a function $h: Y \rightarrow D, Y \subset X$ that for all p_i that we consider for $Y, h(p_i) \in R_i$.

Whether we search the state space completely or incompletely depends on the choice, and hence the properties of a particular algorithm, and especially on the CSP itself. For many problems, it is not usually possible to opt for an algorithm that would search completely (e.g. due to exponential time complexity). Depending on the nature of the CSP, we can use an algorithm that assigns values from the relevant domains (a so-called *initial solution*) randomly or by certain rules to all the variables that we work with in the CSP, so that all the solution constraints are met (or, by some rules, generating assignments to variables until the constraints are met). Alternatively, the solution is designed, which is the most common CSP solution. Constructive algorithms seek solutions based on simple deterministic rules.

Any CSP can be solved by trying all the options. This method is called *generate and test* when every possible combination of values from the domain for the problem variables is systematically generated and tested until all constraints are met. The first combination of values that fits all the constraints of the solution is the solution. The number of combinations we need to go through is the number of Cartesian product elements of the domains of all variables. The method is very inefficient and computationally difficult, so it can be used "at most" for P class problems. The time complexity of the algorithm is always exponential, and it searches the state space completely.

A more successful method is a recursive search (so-called *BACKTRACKING*) [15], which is a certain approach to a range of difficult issues and is generally applicable to a CSP.

The formal algorithm form is as follows:

1. Let us arrange the problem variables arbitrarily into the sequence $x_1, x_2, \dots, x_{|X|}$
2. Let us arrange the domain values arbitrarily into the sequence $d_1, d_2, \dots, d_{|D|}$
3. Call *BACKTRACKING*(1)

BACKTRACKING(i)
if $i > |X|$, *then*
return assignment to variables;
else
for $j = 1,2, \dots, |D|$ *do*
assign value d_j *to variable* x_i*;*
if the assignment to variables $x_1, \dots, x_i$ *is a partial solution, then*
call BACKTRACKING(i + 1);

BACKTRACKING checks for the constraints to be met whenever the algorithm assigns values to all the variables considered at that time. The algorithm is recursive and consists of examining the finite number of subtasks. The process of finding a solution can be seen as a process of repeating attempts to gradually build a state space search tree that usually grows exponentially. However, the time it takes to find the solution depends on:

1. the details of a specific instance of the class of problems
2. the arrangement of problem variables
3. the arrangement of domain values

In some cases, the *BACKTRACKING*(*i*) search procedure can find a solution without performing a backward step, so the time complexity of the problem would only be proportional to the number of variables in the input sequence. *BACKTRACKING* searches the state space completely and chronologically (often referred to as a *recursive chronological search*).

Nowadays, the effort of people involved in the creation of algorithms is to improve the "standard" recursive search, mainly for the purpose of its acceleration. Essentially, it attempts to achieve this acceleration by using extra problem information to search efficiently, thus making fewer unnecessary value assignments from variable domains; that is, when searching, the two variables can be re-assigned the values that have been tested and do not match the constraints. In this way, some parts (branches) of the state space would be searched unnecessarily. Some heuristics make it possible to adapt to new conditions by changing the algorithm, thus finding a solution differently. This is because not all the information may be available to solve various real problems in the initial state of the solution. These are gradually acquired over the course of the solution, and their accumulation can lead to a situation that requires reformulation of the original problem, thus addressing a completely new problem. Using different heuristics reduces the search tree, reducing the complexity of time-consuming exponential complexity. However, just like in *BACKTRACKING*, the computational complexity of the algorithm is exponential. Although this is less common than in standard searching, we cannot predict the actual computational complexity of the problem instance solution nor its average estimate. Therefore, the least

desirable alternative is again considered.

4. Timetable Problem as an Optimization Problem

A *Timetable Problem* (TTP) is another specific case of an $FCSP$, and it requires some combinatorial and optimization capabilities. It is based on different information that respects itself through different constraints. The solution to the problem is usually a "table" consisting of time units and days of the week. Especially for school timetables [16], the time unit is a lesson that consists of a teacher, a room, and a study group. Such a table serves as a predetermined plan to guide the pedagogical process that must be built to resist minor changes, such as a visit to the school that causes a study group to move to another room.

Different requirements placed on the timetable are labelled *necessary* or *complementary*. Necessary requirements must always be met and define the characteristics of the timetable. Without them, the timetable would not be applicable (e.g. it is not possible for the teacher to be in two different places at the same time). Additional requirements may not always be met, although they must be respected as much as possible. They make the timetable more acceptable (e.g. the teacher does not have to teach in the morning and in the afternoon as well). Essential requirements for the timetable are the teachers' (usually when they can or cannot teach), the study groups' (to be at all available), or a requirement for non-compliance (such as the number of rooms). Similarly, the requirement that two classes do not have the same teacher, room or study group is applied as necessary. However, the allocation of the necessary and additional requirements depends on the timetable creator. A common supplementary requirement is the requirement of a special room in which teaching is to take place. Thus, in addition to choosing the right algorithm, the creator of the timetable determines the solution constraints (modelling the problem) according to what timetable he or she creates (for what purpose, for what type of school, etc.).

Achieving the goal depends largely on the algorithm's ability to find the most appropriate solution to the problem from all of its solutions. In order for the algorithm to make the right decision about the solution, it is necessary to set appropriate constraints and criteria for decision making. Based on the previous sections, we can conclude that it is not possible to find a sufficiently fast algorithm to solve a general TTP which is NP-complete. However, the speed itself depends on the nature of the TTP being solved, because for some problem instances the same algorithm can work in polynomial time and for others in exponential time. Furthermore, the success of the TTP solution is also based on selecting the right algorithm according to specific TTP requirements. It may happen that a certain "super" efficient algorithm is well managed for a difficult general TTP, but easy instances of the problem are solved more slowly than when solved by another, less efficient algorithm.

However, the successful processing of easy TTPs is usually possible using any algorithm. The value of the polynomial is still "acceptable" even for a larger input range, and the calculation would in principle always be manageable. However, to find the optimal solution for general difficult TTPs, we need an algorithm with the growth rate of a complexity function exponentially, and therefore the correct formulation of the problem is inevitable (otherwise we would look for a timetable "for years"). Although current technology has not yet exhausted all the possibilities of accelerating calculations by increasing the technical parameters of computers, there are still physical boundaries that do not allow unlimited computing speed. In addition, increasing computer parameters will affect the acceleration of the calculation to a lesser extent than applying an effective algorithm to an efficiently formulated problem. In the case of exponential time complexity algorithms, any computer acceleration will not fundamentally help us (perhaps just the solution on a large number of computers at the same time, but in that case, the TTP model is very complicated to find because of complex communication rules). The right system coupled with the right solution-finding principle can be designed by appropriately defining the problem. It is possible to reduce the number of attempts needed to find a solution (or finding out that the solution does not exist), being the constraints that offer this possibility.

Let us focus our attention on school timetables. A TTP is an optimization problem [6], [7], and therefore the problem is solved by searching the appropriate state space. To find a TTP solution, we set out to:

1. find some (unoptimized) solution
2. find an optimal (complete) solution
3. find a good (partially optimized) solution

It is virtually impossible to find the optimal solution for a general TTP (because of the nature of the problem) while meeting all the necessary and additional requirements. Moreover, it is very difficult (or even impossible) to decide whether the optimal solution found is the best one (is it better to teach maths on Mondays or Tuesdays if it is possible on either day?). Mainly (in formulating a larger-scale TTP), it is not even possible to choose an algorithm that searches the entire state space because of time complexity. That is why we will be satisfied with a good solution meeting all the necessary and some additional requirements. Only finding a solution which is possible in the shortest time when just the necessary requirements are met is not a very desirable condition. With that timeframe, we can try to adjust the input word of the problem if possible (e.g. choose different rooms for certain subjects or otherwise specify some constraints). It is necessary to characterize the input

differently when finding an incomplete TTP solution. Our TTP model is formulated only with the necessary requirements, so the solution we are looking for is complete.

As mentioned, a TTP is an NP-complete problem. One way to show this is to find a problem that we know is NP-complete and is an equivalent problem to a TTP (or is of the NP class and can be reduced). Such a problem is k-colourability of the non-oriented graph [1] [17]. In the graph model, the vertex represents a class and the edge stands for a time slot. Two classes that have some resources in common (where the resources mean teachers, rooms, etc.) are displayed at such vertices of the graph that are bound by the edge. Class scheduling for time slots is then controlled by vertex colouring [18].

Let us consider n sets of rectangles so that for $\forall i, 1 \leq i \leq n$ we have a set of beginnings $X_i \times Y_i$ and a set of sizes $W_i \times H_i$. The task now is to select n rectangles so that they do not overlap in pairs. More specifically, for $\forall i, 1 \leq i \leq n$ it is necessary to find the beginning $(x_i, y_i) \in X_i \times Y_i$ and the size $(w_i, h_i) \in W_i \times H_i$, so that:

$$\forall 1 \leq i < j \leq n\big(x_i + w_i \leq x_j\big) \vee \big(x_j + w_j \leq x_i\big) \vee \big(y_i + h_i \leq y_j\big) \vee \big(y_j + h_j \leq y_i\big).$$

The polynomial reducibility of the problem, that is "*Sequencing with release times and deadlines*," shows that such a selection of rectangles is an NP-complete problem [19]. Hence, the rectangle problem is a TTP.

5. Universal Formulation of a TTP as a $FCSP$

A *time slot* is the time in the timetable that we assign to a unit class. We assume that there is a constant length of break time (which we do not pay attention to) assigned to time slots. We do not assume that teaching is carried out at remote locations, and therefore there is no need for more time to move between classes. A *block of classes* is made up of two consecutive time slots which are not separated by a break (this is generally the most common case in schools) and are assigned the same resources.

Let the number of teaching days and the number of time slots (not necessarily the same for each day) be given in advance (e.g. by school management). Let us also assume that the school determines the subjects taught and assigns the teachers to the individual subjects. Next, let any combination of subjects be admissible for students who are assigned to study groups. We assume that students are available for class at any time.

If the number of students choosing the subject is large, we will assign students into groups that we call *subject groups*. We further assume that all groups for the same subject are in class the same number of hours. Assigning a student to a subject group means assigning him or her to all the classes of the subject. It is thus sufficient to assign the student only to his or her study groups.

A TTP is then characterized by the input word V_1, which consists of input of time, input of teachers, input of rooms, input of subjects, input of study groups and input of classes. All input sets are assumed to be non-empty. $V_C = (C, D, D_C, BH)$ characterizes the input of time where C is the finite set of time slots, D is the finite set of days, $d_C : C \longrightarrow D$ where $d_C(c)$ denotes the day of the time slot $c \in C$ and $BH : C \times C \to B$, where $B = \{FALSE, TRUE\} \Leftrightarrow \{0,1\}$ is the set of truth-values, and function $BH(c_1, c_2)$ expresses whether two time slots $c_1, c_2 \in C$ are a block of classes. The relation BH is symmetric because $\forall c_1, c_2 \in C: BH(c_1, c_2) \Leftrightarrow BH(c_2, c_1)$. But it is not reflexive because $\forall c \in C: BH(c, c) \Leftrightarrow 0$. Pair $V_U = (U, C_U)$ characterizes the input of teachers, where U is the finite set of teachers and $C_U : U \to P(C)$, where $P(C)$ is the potent set of the set C and $C_U(u)$ denotes the set of time slots when the teacher $u \in U$ is available. $V_M = (M, T, t_M)$ characterizes the input of rooms, where M is the finite set of rooms, T is the finite set of room types and $t_M : M \to T$, where $t_M(m)$ denotes the room type $m \in M$. $V_P = \big(P, R, Q, p_Q, r_P, max\big(H_Q\big), max(S_P)\big)$ characterizes the input of subjects, where P is the finite set of subjects; R is the finite set of class years; Q is the finite set of subject groups; $p_Q : Q \to P$, where $p_Q(q)$ denotes the subject of the subject group $q \in Q$; $r_P : P \to R$, where $r_P(p)$ denotes the year in which the subject $p \in P$ is taught; $max\big(H_Q\big) : Q \to N$, where $max\big(H_Q(q)\big)$ denotes the upper limit of the number of hours of subject group $q \in Q$ that can be taught on the same day, N is the set of natural numbers; and $max(S_P) : P \to N$, where $max\big(S_P(p)\big)$ denotes the maximum number of students who can be assigned to any subject group $p \in P$.

According to their study programme, we put together individual students in study groups. All students of the study group attend classes together. The study group is made up of at least one student and, at the most, of all students attending the school. $V_K = (K, P_K, r_K, n_K)$ characterizes the input of study groups, where K is the finite set of study groups; $P_K : K \longrightarrow P(P)$, where $P_K(k)$ denotes the set of subjects chosen by each student from study group $k \in K$; $r_K : K \to R$, where $r_K(k)$ denotes the year of the students from study group $k \in K$; and $n_K : K \to N$, where $n_K(k)$ denotes the number of students in study group $k \in K$. $V_H = (H, T_H, u_H, q_H, BL)$ characterizes the input of teaching classes, where H is the finite set of teaching classes; $T_H : H \longrightarrow P(T)$, where $T_H(h)$ denotes the set of rooms that are suitable for teaching, $h \in H$; $u_H : H \to U$, where $u_H(h)$ denotes the teacher's classes, $h \in H$; $q_H : H \to Q$, where $q_H(h)$ denotes the group for the class subject, $h \in H$; and $BL : H \times H \to B$, where $BL(h_1, h_2)$, expresses whether two hours $h_1, h_2 \in H$ are from the block of hours. The relation BL is symmetric because $\forall h_1, h_2 \in$

$H: BL(h_1, h_2) \Leftrightarrow BH(h_2, h_1)$. But it is not reflexive, because $\forall h_1, h_2 \in H: BL(h, h) \Leftrightarrow 0$. One hour may be with at most one more hour in the block of hours, hence $\forall h, h_1, h_2 \in H: BL(h, h_1) \wedge BL(h, h_2) \Leftrightarrow h_1 = h_2$. In total, $V_1 = (V_C, V_U, V_M, V_P, V_K, V_H)$.

Let us use A to denote the set of proper pairs $(k, p) \in K \times P$, hence $A = \{(k, p) \in K \times P,\ k \in K \wedge p \in P_K(k)\}$. Let function $Q_P : P \to P(Q)$ yield the set of subject groups for each subject, thus for $p \in P\ Q_P(p) = \{q \in Q, p_Q(q) = p\}$ is the set of subject groups. Then, let us assign the study groups to one possible subject. Thus, assigning the subject groups is function $q_A : A \to Q$, where $\forall (k, p) \in A: q_A(k, p) \in Q_P(p)$. The time slot assignment consists of assigning classes to time slots, hence the function $c_H : H \to C$. The room assignment consists of assigning classes to rooms, hence the function $m_H : H \to M$.

Let us set the constraints. Let function $Q_P : P \to P(Q)$ yield the set of subject groups for each subject. For the given time slot assignmentc_H, let function $H_{Q,D} : Q \times D \to P(H)$ yield the set of group classes for the subject assigned by the time slot on a specific day, that is $H_{Q,D}(q, d) = \{h \in H, q_H(h) = q \wedge d_C(c_H(h)) = d\}$. Then, if the upper limit of the number of hours for each day of the group $q \in Q$ for the subject that can be taught on the same day is $max\left(H_Q(q)\right)$, then for $\forall q \in Q\ \forall d \in D$ it must hold true that $\left|H_{Q,D}(q, d)\right| \leq max\left(H_q(q)\right)$. Let c_H be the given time slot assignment. Let $H' \subseteq H$ be the set of classes that share certain resources (rooms, study groups, subject or teacher groups). Then, it must be true that $\forall h_1, h_2 \in H': h_1 \neq h_2 \Rightarrow c_H(h_1) \neq c_H(h_2)$ while none of the two hours sharing the resources can be assigned into the same time slot. Let us denote this constraint as $Resources(H')$. Let m_H be the given room assignment. Let the function $H_M : M \to P(H)$ return for each room $m \in M$ the set of classes $H_M(m) = \{h \in H, m_H(h) = m\}$, assigned to the same room. Then, for $\forall m \in M$ the constraint $Resources(H_M(m))$ must be met. For the given input of classes V_H we will determine the set of classes for each group $q \in Q$ for the subject as function $H_Q : Q \to P(H)$ where $H_Q(q) = \{h \in H, q_H(h) = q\}$. Then, $\forall q \in Q$ and the constraint $Resources\left(H_Q(q)\right)$ must be satisfied.

If pair $(k, p) \in A$ has been assigned to subject group $q = q_A(k, p)$, then study group $k \in K$ must attend each class of set $H_Q(q)$. Let q_A be a given assignment for subject groups. Let $Q_K(k)$ be a set of the subject groups to which the study group $k \in K$ has been assigned, hence $Q_K(k) = \{q \in Q, \exists p \in P_K(k): q_A(k, p) = q\}$. Then, for each study group $k \in K$, the set of classes of study group $k \in K$ is the set of classes to which study group $k \in K$ has been assigned, hence $H_K(k) \bigcup_{q \in Q_K(k)} H_Q(q)$. Then, two different classes of the same study group cannot be assigned to the same time slots, and given that $\forall k \in K$constraint $Resources(H_K(k))$ must be satisfied.

Let $\forall h \in H$ know the teachers of each class, that is, set $u_H(h)$. Then, function $H_U : U \to P(H)$ displays each teacher for the set of classes that he or she teaches, thus for $\forall u \in U$ the set of classes the teacher teaches is $H_U(u) = \{h \in H, u_H(h)u\}$. Then, $\forall u \in U$ and constraint $Resources(H_U(u))$ must be met.

In addition, for teachers, we assume that not all time slots are always available (e.g. they work part-time). Therefore, it is necessary to use such a constraint to make the time slot assignment impossible for the time slot when he or she is not available. Let c_H be the given time slot assignment. Then, this assignment matches the teacher's availability, and if $\forall u \in U$ and $\forall h \in H_U(u)$, it holds true that $c_H(h) \in C_U(u_H(h))$.

In most schools, some rooms are designed for teaching special subjects (e.g. laboratories and gyms). In that case, certain subjects can only be taught in these special rooms. This is ensured by the following constraint: let m_H be the given room assignment. Then, $\forall m \in M$ denotes room type $t_M m$. Set $T_H(h)$ is the set (of types) of matching rooms for class $h \in H$. Then, for $\forall h \in H$ it must be true that $t_M(m_H(h)) \in T_H(h)$.

The capacity of the subject groups is limited, and we want the subject groups for the same subject to be approximately the same size. Thus, all groups $q \in Q_P(p)$ for subject $p \in P$ cannot exceed a certain number of students. This is ensured by the following constraint: let q_A be the given group assignment for the subject. Then, $\forall q \in Q$ and for the corresponding subject $p = p_Q(q)$ the set $K_Q(q) = \left\{k \in K, p_Q(q) \in P_K(k) \wedge q_A\left(k, p_Q(q)\right) = q\right\}$ is the set of the study groups assigned to group $q \in Q$ for the subject. Let $n_K(k)$ denote the size of study group $k \in K$. Then, assigning q_A to the subject groups matches the upper limit of the group size for the subject. If $\forall p \in P$, $\forall q \in Q_P(p)$, the maximum number of students of the group for the subject $\forall q \in Q_P(p)$ is less than or equal to the given upper limit $max(S_P(p))$. Thus, it must hold true that $\sum_{k \in K_Q(q)} n_K(k) \leq max(S_P(p))$.

Furthermore, it is necessary to set constraints for hour blocks that exclude those TTP solutions that would not meet them. Two hours in the block must be assigned directly to consecutive time slots and the same room. The given time slot assignment c_H matches the block of hours only if $\forall h_1, h_2: BL(h_1, h_2) \Rightarrow BH(c_H(h_1), c_H(h_2))$. The given room assignment m_H matches the block of hours if and only if $\forall h_1, h_2, BL(h_1, h_2) \Rightarrow m_H(h_1) = m_H(h_2)$.

Then, $V_2 = (q_A, c_H, m_H)$ is the complete TTP solution if all assignments to all the variables considered are consistent, so all the constraints are met. If not all variables have an assignment but the constraints are met, the solution is called partial. Our task is to find a complete solution without evaluating the solution.

To find a solution, we will use a modified *tree-search algorithm* [16] [20] that can handle a large number of

general TTP instances. For ease of presentation (though not affecting the behaviour of the algorithm), let us simplify the solution by not considering room assignment m_H nor the constraint of room type t_M. However, let us consider the constraint of the availability of rooms. Thus, for each instance of $c \in C$, the number of hours $|H_C(c)|$ allocated by time slots where $H_C(c) = \{h \in H, c_H(h) = c\}$ cannot exceed the number of rooms $|M|$. Next, let us not consider the blocks nor the constraints placed on the blocks. We will focus on looking for pair (q_A, c_H) so that the time slot assignment (a so-called *timetable*) and the assignment to subject groups is complete (denoting $Comp(q_A, c_H)$, or possibly $Comp(q_A)$, $Comp(c_H)$) so that each pair $(k,p) \in A$ is assigned to the appropriate subject group $q \in Q_P(p)$ and that each class has a time slot). Then, the TTP solution is complete if $Comp(q_A) \wedge Comp(c_H)$.

6. Algorithm and Implementation

First, we will schematically describe our problem:

Domains:

set of years R, set of teachers U, set of study groups K, set of subject groups Q, set of hours H, set of time slots C, set of subjects P, set of days of the week D

Interpretation of variables:

$$r_K : K \to R,\ r_P : P \to R,\ u_H : H \to U,\ q_H : H \to Q,\ Q_P : P \to P(Q),\ C_U : U \to P(C),\ P_K : K \to P(P),\ d_C : C \to D,\ n_K : K \to N$$

Set limits:

$$|M| \in N,\ max(H_Q): Q \to N,\ max(S_P): P \to N$$

Objective:

find $q_A : A \to Q$, $c_H : H \to C$ so that all the constraints are met and that $\forall (k,p) \in A: q_A(k,p) \in Q_P(p)$, where $A = \{(k,p) \in K \times P, k \in K \wedge p \in P_K(k)\}$

Constraints:

$$\forall q \in Q\ \forall d \in D: |H_{Q,D}(q,d)| \le max\left(H_q(q)\right)$$

(denoted (o_1))

$$\forall c \in C: |H_C(c)| \le |M|$$

(denoted (o_2))

$$\forall k \in K: Resources(H_K(k))$$

(denoted (o_3))

$$\forall q \in Q: Resources\left(H_Q(q)\right)$$

(denoted (o_4))

$$\forall u \in U: Resources(H_U(u))$$

(denoted (o_5))

$$\forall p \in P\ \forall q \in Q_P(p): \sum_{k \in K_Q(q)} n_K(k) \le max(S_P(p))$$

(denoted (o_6))

$$\forall u \in U\ \forall h \in H_U(u): c_H(h) \in C_U(u_H(h))$$

(denoted (o_7))

The main idea of the algorithm is that the assignment to subject groups is sequentially realized by always assigning only one pair $(k,p) \in A$, so that the consistency of all variables is assured. We say that the solution is consistent (denoted $Cons(q_A, c_H)$, or $Cons(q_A)$, $Cons(c_H)$) if all the constraints are met, respectively if the assignments q_A, c_H meet all the constraints (denoted $Meet(q_A, c_H, O)$, where O is the set of solution constraints). Then, $Cons(q_A, c_H) \Leftrightarrow Meet(q_A, c_H, O)$. The variables representing solution (q_A, c_H) and sets $Q_A(a)$ of the admissible subject group pairs $a \in A$ are declared globally.

At the beginning, there is no assigned pair $a = (k,p) \in A$. Thus, we will assign a pair to one of the allowable subject groups, $q \in Q_A \in (a) \subseteq Q_P(p)$, so that all the constraints imposed on the solution are met. We will get a consistent partial solution (q_A, c_H), and it holds true that $Comp(c_H)$. This is how we will proceed, by always trying to consistently extend the last assignment q_A to the subject groups. Whether $Cons(q_A \cup \{(a,q)\}, c_H)$ is true can be decided in polynomial time [16]. We will always align admissible subject groups and pairs using *RANDOM* heuristics [21]. We will select them one by one until there is a consistent extension. If new assignment $q'_A = q_A \cup \{(a,q)\}$ to the subject groups meets all the group constraints, we will have to decide whether current timetable c_H is also consistent for q'_A. If timetable c_H is not consistent and thus at least one hour $h_1 \in H_K(k)$ of study group $k \in K$ is in conflict with hour $h_2 \in H_Q(q)$ of subject group $q \in Q$, it is possible that another timetable, c'_H, is consistent for q'_A and needs to be found (if any). The recommended search method is a so-called *tabu search* algorithm [21]. The set of conflicts (the only constraints that can be unfulfilled are o_3, o_4, o_5). in terms of new solution resources will be denoted $Conf(q'_A, c_H)$, that is $h_1, h_2 \in Conf(q'_A, c_H)$.

Formally, this procedure can be written as follows:

PAIR ASSIGNMENT(pair a)
$(q_0, \dots, q_{j-1})$ *= ordered sequence* j *of subject groups in* $Q_A(a)$ *according to the RANDOM rule;*
i = 0;
result = FALSE;
if (result = FALSE) and (i < j) do
$q'_A = q_A \cup \{(a, q_i)\}$*;*
if $Meet(q'_A, o_6)$ *and* $|Conf(q'_A, c_H)| > 0$*,then*
c'_H *= FIND THE TIMETABLE SOLUTION;*
if $|Conf(q'_A, c_H)| = 0$*, then*

$c_H = c'_H$;
if $Conf(q'_A, c_H)$, *then*
$q_A = q'_A$;
result = TRUE;
else
$Q_A(a) = Q_A(a) - \{q_i\}$;
i = i + 1;
return result;

If all allowable subject groups for pair $a = (k, p)$ have been tested and solution (c_A, c_H) cannot be extended so that timetable c_H is consistent, the algorithm has found only an incomplete solution (it has failed). One of the reasons may be that there is no complete solution, but the more likely reason is that no consistent complete solution has the current assignment to the subject groups as its partial solution, i.e.

$$\left(\exists {q'}_A : A \to Q\ \exists {c'}_H : H \to C\left(q_A \not\subseteq {q'}_A\right) \wedge Comp({q'}_A, {c'}_H) \wedge Cons({q'}_A, {c'}_H)\right) \Leftrightarrow 0.$$

Then, at least one of the previous assignments must be changed. We could start looking for a solution from scratch, which (because of the *RANDOM* rule) would start looking for another TTP solution, and so we may be able to avoid failure. In the case of this procedure, it is possible to start another search by remembering the failure status $\left((q_A, c_H), a, Q_A\right)$ of the algorithm and by influencing the next attempt accordingly. For example, we could start by assigning just the conflicting pair or placing it at the beginning of the sequence and the other pairs according to the *RANDOM* rule. However, *BACKTRACKING* is a better way to go back to the penultimate assignment and try to take into account another acceptable subject group (the next one in order). Let a be a pair be where the algorithm failed. Then, the pair that was assigned as the previous one is denoted $pred(a)$, and $q = q_A\left(pred(a)\right)$ denotes the subject group to which $pred(a)$ was assigned. We will remove $\{pred(a), q\}$ from current subject group assignment q_A. Then, since timetable c_H remains consistent, we can continue finding a solution to partial solution $(q_A - \{pred(a), q\}, c_H)$. When selecting *BACKTRACKING*, it is not necessary to remember the failure status. Ordered sets Q_A and the order in which the previous pairs have been assigned represent a so-called *tree of partial assignments* to subject groups (that is why the algorithm is also called tree-search). Since reassigning $pred(a)$ to group k did not make sense, we can remove k from allowable subject group domain $Q_A\left(pred(a)\right)$ (which generally reduces the effort of *BACKTRACKING*). Next, at the point of failure, set $Q_A(a)$ for pair $a = (k, p)$ is empty, and therefore, before performing the backward step, it is necessary to reinitialize set $Q_A(a)$.

The complete algorithm can be formally written as follows:

TREE-SEARCH ALGORITHM(TTP)
i = 0;
repeat
A' *= set of all pairs ordered by the RANDOM rule;*
for all $(k, p) \in A'$ *perform*
$Q_A(k, p) = Q_P(p)$;
restart = FALSE;
if $A' \neq \emptyset$ *and (restart = FALSE) do*
a = next pair from A';
result = PAIR ASSIGNMENT(a);
if the result, then
$A' = A' - \{a\}$;
or else
if perform BACKTRACKING, then
$q_A = q_A - \{(pred(a), q)\}$;
$Q_A\left(pred(a)\right) = Q_A\left(pred(a)\right) - \{q\}$;
$Q_A(a) = Q_P(p)$;
$A' = A' \cup \{pred(a)\}$;
else
restart = TRUE;
reinitialize (q_A, c_H);
i = i + 1;
until ($A' \subseteq$ *sets of already assigned pairs* q_A*) or (i >* *allowed number of restarts);*
return $A' \subseteq$ *sets of already assigned pairs* q_A*;*

To assign a pair, there must be determined *uninterruptibility* requirements (that is, until a specific pair is assigned, the algorithm will not start working with another one) as well as *time independence* (the time required to assign a pair, a so-called *process time*, which is not limited, although it is assumed that for each pair it can be any length of time – to find a solution to timetable ${c'}_H$).

We implemented the algorithm into the program:

Intended input: C, D, U, M, P, Q, K, H (no blocks)
Intended assignments: $m_H,\ q_A,\ c_H$
Intended constraints: $Resources\left(H_M(m)\right)$,$o_3$,$o_4$,$o_5$

And the algorithm's behaviour has been tested under these conditions (input data):

$|D| = 4$ and set C is determined by the map (Figure 1)

	1	2	3	4	5	6	7
Monday	✓	✓	✗	✓	✓	✓	✗
Tuesday	✓	✓	✗	✓	✓	✓	✗
Wednesday	✓	✓	✓	✓	✓	✓	✗
Thursday	✓	✓	✗	✗	✓	✓	✗
Friday	✗	✗	✗	✗	✗	✗	✗
Saturday	✗	✗	✗	✗	✗	✗	✗
Sunday	✗	✗	✗	✗	✗	✗	✗

Figure 1. Set of time slots

$|U| = 4$ (teachers U_1, U_2, U_3,U_4)

$|K| = 4$ (study groups K_1, K_2, K_3,K_4), where each group has to complete 20 classes ($|P| = 80$) and the timetable is fully covered (all from subjects P_1, P_2, P_3,P_4 are in the range of 5 hours)

$|M| = 4$ (rooms M_1, M_2, M_3,M_4)

Subject P_i is taught in room M_i by teacher U_i for $i = 1, \dots, 4$.

The number of complete solution search observations $V_2 = (q_A, c_H, m_H)$ (in the programme called *timetable generation*): 5.

The time complexity that is polynomial for this (simplified) model [16] has been reflected in the rate of generation in our created programme. The generation process is captured on the graphs as follows (Figure 2 – Figure 6).

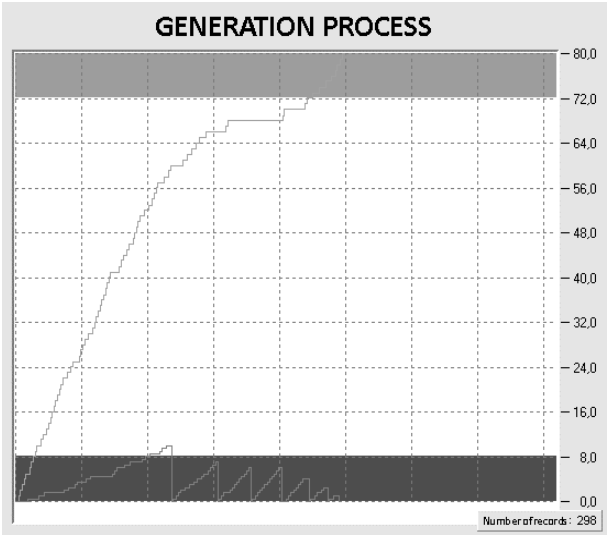

Figure 2. First observation

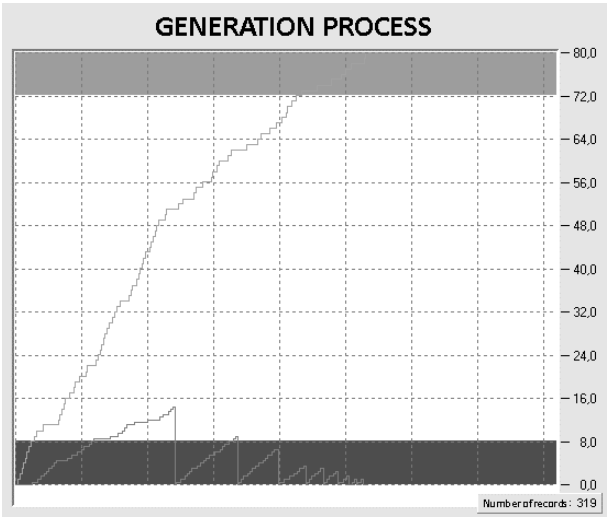

Figure 3. Second observation

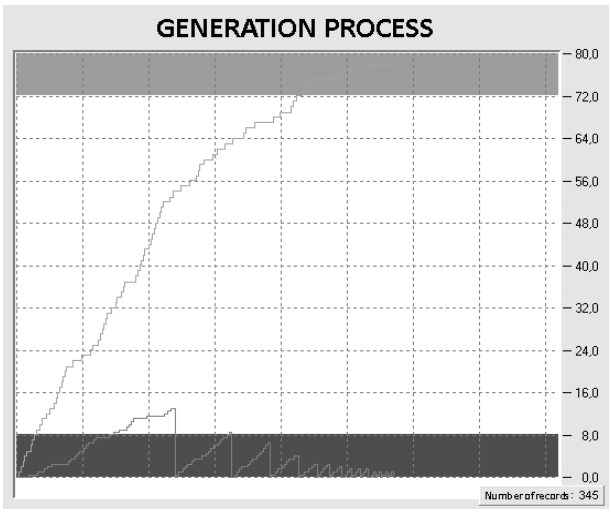

Figure 4. Third observation

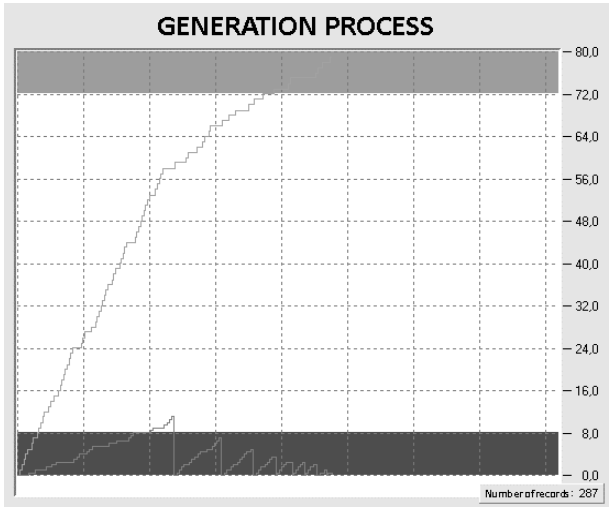

Figure 5. Fourth observation

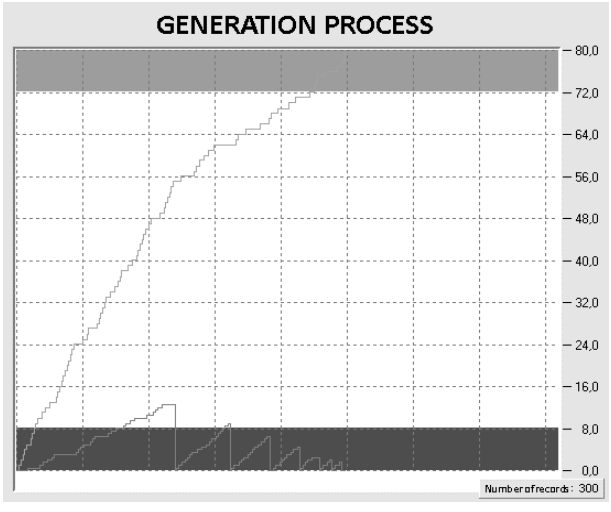

Figure 6. Fifth observation

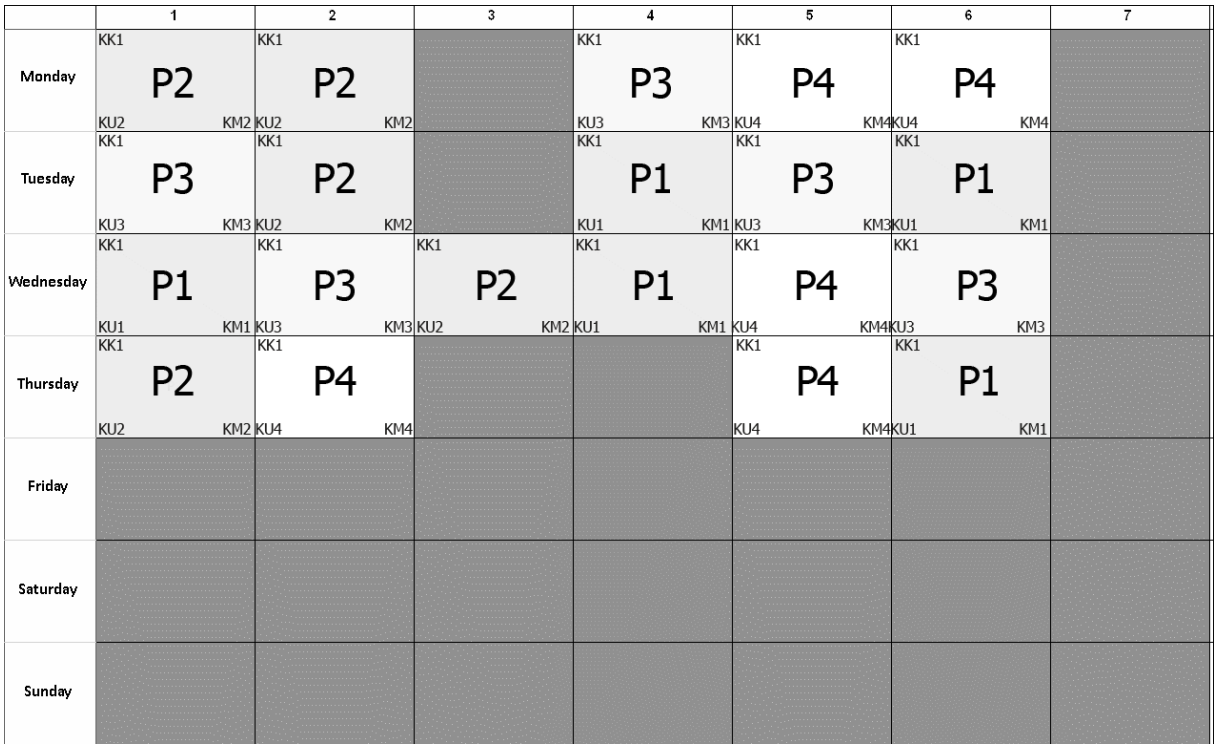

Figure 7. A generated timetable example for study group K_1

A complete solution (Figure 7) has always been found up to 30s (on average in 27.6s). From the graphs listed in the annex, it can be seen that the generation had approximately the same pattern (the algorithm behaved "similarly" despite the *RANDOM* heuristics used).

7. Conclusions

The aim of the paper was to find a suitable mathematical formulation of the Timetable Problem, which is modelling the Timetable Problem as a Constraint Satisfaction Problem with reference to the general possibilities of algorithmic solution. Then we verified the model algorithmically and experimentally in the created program. From this point of view, the work is largely devoted to algorithms as an effective problem-solving tool. Despite the fact that we are able to solve (and effectively solve) a number of problems by way of the Constraint Satisfaction Problem, this notion is not well-known, and the constraint options are not exhausting. The theory of constraints (dealing with problems and their properties, simplifying problems or effective ways of solving them) as a separate theory is currently being originated (although its content is familiar to mathematicians). There is no literature completely describing the properties and relationships to enable us to express any problem as the Constraint Satisfaction Problem (or simplify it) so that we can solve it or at least decide on the solution. Perhaps the greatest knowledge is contained in Constraint Logic Programming or Artificial Intelligence. However, the constraints are sure to be the right way to model and work out problems. Even the different challenging issues, if properly mathematically formulated, can be easily solved algorithmically.

Conflicts of Interest

The authors declare that they have no conflicts of interests.

REFERENCES

[1] Aho A. V., Hopcroft J. E., Ullman J. D., "The Design and Analysis of Computer Algorithms", Massachusetts, Addison-Wesley Publishing Co., 1974, ISBN 0-20100029-6.

[2] Jeavons P., Cohen D., Gyssens M., "How to determine the expressive power of constraints", In: Constraints: An International Journal, Kluwer Academic Publishers, pp. 1-19, 1998, Boston.

[3] Bistarelli S., Montanari U., Rossi F., Schiex T., Verfaillie G., Fargier H., "Semiring-based CSPs and Valued CSPs: Frameworks, Properties, and Comparison", In: Constraints: An International Journal, Kluwer Academic Publishers, pp. 275-316, 1999.

[4] Wirth N., "Algorithms and data structures", 2nd ed., Bratislava: Alfa, 1989, ISBN 80-05-00153-3.

[5] Stone H. S., Stone J., "Efficient Search Techniques: An Empirical Study of the N-Queens Problem", In: Technical Report RC 12057, New York, IBM T. J. Watson Research Center, 1986.

[6] Volná E., "Evolutionary algorithms and neural networks", Ostrava: University in Ostrava, PROJECT: CZ.1.07/2.2.00/28.0245, EDUCATION FOR COMPETITI VENESS, 2012.

[7] Míka S., "Mathematical optimization", Textbook, Plzeň: ZČU, 1997.

[8] Jeavons P., "On The Algebraic Structure Of Combinatorial Problems", In: Theoretical Computer Science, Elsevier, Vol. 200, Issues 1-2, pp. 185-204, 1998.

[9] Jeavons P., Cohen D., Cooper M., "Constraints, Consistency, and Closure", In: Artificial Intelligence, Elsevier, Vol. 101, pp. 251-265, 1998.

[10] Jeavons P., Cooper M., "Tractable Constraints on Ordered Domains", In: Artificial Intelligence. Elsevier, Vol. 79, pp. 327-339, 1995.

[11] Jeavons P., Cohen D., Gyssens M., "Closure Properties of Constraints", In: Journal of the ACM. Association for Computing Machinery, Vol. 44, No. 4., pp. 527-548, 1997.

[12] Cohen D., Jeavons P., Koubarakis M., "Tractable disjunctive constraints", In: Principles and Practice of Constraint Programming-CP97, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Vol. 1330, 1997.

[13] Kumar V., "Algorithms for Constraint Satisfaction Problems", In: AI Magazine, Vol. 13, No. 1, pp. 32-44, 1992, DOI:10.1609/aimag.v13i1.976.

[14] Yokoo M., Durfee E. H., Ishida T., Kuwabara K., "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms", In: IEEE Transactions on Knowledge and DATA Engineering, Vol. 10, No. 5, pp. 673-685, 1998.

[15] Bitner J., Reingold E. M., "Backtrack Programming Techniques", In: Communications of the ACM, Vol. 18, No. 11, pp. 651-655, 1975, DOI: 10.1145/361219.361224.

[16] Willemen R. J., "School timetable construction: Algorithms and complexity", Eindhoven: TUE, ISBN 90-386-1011-4, 2002, DOI:10.6100/IR553569.

[17] Redl T. A., "A Study of University Timetabling that Blends Graph Coloring with the Satisfaction of Various Essential and Preferential Conditions", PhD. thesis, Houston, Texas, Rice University, 2004.

[18] Neufeld G. A., Tartar J., "Graph coloring conditions for the existence of solutions to the timetable problem", In: Communications of the ACM, Vol. 17, No. 8, pp. 450-453, 1974, DOI: 10.1145/361082.361092.

[19] Garey M. R., Johnson D. S., "Computers and Intractability – A Guide to the Theory of NP-Completeness", New York, USA, W. H. Freeman & Co., 1990, ISBN 0716710455.

[20] Haralick R., Elliot G., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", In: Artificial Intelligence, Vol. 14, No. 3, pp. 263-313, 1980, DOI:10.1016/0004-3702(80)90051-X.

[21] Schmotzer M., "Analyze and design new tools for solving time scheduling tasks in the boundary logic programming environment", Diploma thesis, Košice, FEI TU, 1997.