Solving the High School Scheduling Problem Modelled with Constraints Satisfaction using Hybrid Heuristic Algorithms

Ivan Chorbev, Suzana Loskovska, Ivica Dimitrovski and Dragan Mihajlov
Faculty of Electrical Engineering and Information Technologies
Republic of Macedonia

1. Introduction

Constraint Programming is a methodology for problem solving which allows the user to describe data and constraints of the problem without explicitly solving in the declarative phase. Constraint Satisfaction Problems (CSP) can simply be defined as a set of variables and a set of constraints among the values of the variables. Typical method of solving CSP models is building the solution by backtracking approach in which a partial assignment to the variables is incrementally extended, while maintaining feasibility of the current solution. The constraints are kept satisfied throughout the solving process.

Many optimization problems of practical as well as theoretical significance consist of finding "the best" configuration of values for a set of variables. Such problems where the solution is modelled using discrete variables belong to combinatorial optimization (CO). The problems of combinatorial optimization consist of a set of variables, their domains, constraints among variables and a goal function that requires to be optimized. School scheduling is a typical example of a CO problem.

High school schedule generation includes both temporal and spatial scheduling. It is a computation demanding and usually a complex task. It is a NP hard optimization problem that requires a heuristic solving approach (Zhaohui & Lim, 2000).

It is interesting to note that educational institutions rarely use automated tools for schedule generation, although the area has been researched for a long time. A survey in British universities (Zervoudakis 2001) showed that only 21% of the universities use a computer in the generation of exam timetables. Only 37% of the universities use the computer as assistance in the process, while 42% do not use a computer at all. Generation of schedules in some schools in Japan takes up to 100 man hours a year. In bigger schools, schedule generation begins in April and does not end until June, two months after the beginning of the school year, almost 150 work days.

Constraint satisfaction is usually not the first choice for modelling scheduling problems, due to their high complexity. Only the final schedule (hopefully) satisfies all imposed constraints. During schedule generation, most of the constraints will be dissatisfied at some point. We created a system where the extent of constraint satisfaction is measured and compared, so CSP can be successfully used in scheduling (Chorbev et al. 2007). When a measurement of constraint satisfaction is included, the system becomes a Constraint Optimization Problem (COP).

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

A very general approach to solve combinatorial optimization problems like scheduling is to generate an initial, suboptimal solution and then to apply heuristics to improve the solution. This method is somewhat incompatible with the standard backtracking method in constraints programming. In NP hard problems, building the solution by backtracking approach in which a partial assignment to the variables is incrementally extended is virtually impossible. For example, when the number of queens in the N-Queens problem exceeds certain number (for example 50), the Generalised Arc Consistency – Conflict Based Back-jumping (GAC-CBJ) algorithm fails to give a result in reasonable time (Jolevski et al., 2005b). The duration of the search quickly grows beyond any reasonable amount.

Initially, the aim of our research was to achieve a successful symbiosis of constraint programming and heuristic algorithms. Additionally, we aimed to create hybrid heuristic algorithms that would use the advantages of known algorithms. Therefore, we developed hybrid combinations of different heuristic approaches. Our solving approach begins with an initial suboptimal solution followed by heuristic repair to achieve the final correct solution. Ideas from algorithms like Simulated Annealing, Tabu Search, and Guided Search were incorporated to achieve quicker and more accurate solving. Heuristic algorithms demand a constraint satisfaction system that can measure the level of constraint satisfaction and provide heuristics for solution improvement (Leenen et al. 2003).

Most of the heuristic decisions in our solving process are made in the process of repairing the first suboptimal solution. We had to implement mechanisms to avoid trapping at local optima, avoid deadlocks and achieve convergence. The functions that generate the proper next solution based on the previous one are the key to successful, quick and accurate solving. They use knowledge about the problem and reuse information of the specific inconsistent constraints to generate an improved solution in the neighbourhood of the current one. Exact generation of improved solutions based on previous inconsistencies showed useful up to the moment when a deadlock occurs. A stochastic component in the solving process proved effective in avoiding deadlocks and guiding toward the final solution.

To test and implement the hybrid algorithms and their use over constraint modelled problems, a broader software framework was necessary. Therefore we developed and implemented a universal Constraint Solving Engine (CSE) and a Constraint Programming Library (CPL). We developed a set of constraint types for modelling different problem types and a mechanism for selection an optimal algorithm for the given problem. CPL contains different algorithms, including Simulated Annealing, Tabu search, Arc consistency etc., as well as their hybrids.

This approach offers several advantages. The tool can be applied on different problems by cost of very little to no further programming at all. For the user it is only necessary to model a new problem with given constraint types, choose an algorithm, and initiate the solution process. The engine can be easily implemented in any commercial problem solving software. Our solving engine has substantial theoretical implications, too. The use of object-oriented approach provides a mechanism for adding and testing new algorithms based on the same problem description. This system enables comparing efficiency, and results of different algorithms as well.

Part 2 of the chapter gives an overview of Constraint Programming and Constraint Optimization where the origins, the definitions and the basic concepts are explained. Part 3 of the chapter gives an overview of the concept of hybridization of heuristic algorithms. Some strategies are explained and examples are given. Part 4 of the chapter gives a short

description of the constraint solving engine with multiple optimization algorithms that we developed and used for simulations. Part 5 gives the model of the high school scheduling problem expressed in terms of mathematical constraints. The hybrid heuristic algorithm that we developed is explained in part 6 of the chapter. Finally, part 7 of the chapter contains the closing remarks and ideas for future work.

2. Constraint programming and constraint optimization

During the seventies of the 20th century, David Waltz within one of his algorithms set the basic concept of the technique of Constraint Propagation (Kumar, 1992). Ever since, the concept has evolved surpassing the boundaries of artificial intelligence and affecting wide range of research areas. Today, an increasing number of explorers in the area of programming logic, knowledge representation, expert systems, theoretical computer science, operational research, and other similar fields explore the use of constraint programming techniques, both as theoretical basis as well as true practical applications. In time, it is understood that constraint satisfaction is the main problem of a wasp area of problems like time reasoning, spatial planning, configuration planning, timetable generation, telecommunications, even in databases (Der-Rong & Tseng, 2001). The main reason for the increased interest and success of constraints processing techniques is their ability for good declarative formulation of problems as well as efficient solving (Meyer, 1994).
A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain (Bartak, 1999). More formal definition states:

Definition 1: (Gavanelli, 2002)
A Constraint Satisfaction Problem (CSP) is a triple P = {X, D, C} where:
$X= \{X_1, X_2, ..., X_n\}$ is a set of unknown variables,
$D = \{D_1, D_2, ..., D_n\}$ is a set of domains and
$C = \{c_1, c_2, ..., c_n\}$ is a set of constraints.
Each $c_1(X_{i1}, ..., X_{ik})$ is a relation, i.e., a subset of the Cartesian product $D_{i1} \times \cdots \times D_{ik}$.
An assignment $A=\{X_1 \rightarrow d_1, \ldots, X_n \rightarrow d_n\}$ (where $d_1 \in D_1, \ldots, d_1 \in D_{n1}$) is a solution if it satisfies all constraints.

In the declaration phase of Constraint Programming, the user describes the data and the constraints of the problem without explicitly solving it. When using constraints, the problems can simply be defined with a set of variables and a set of constraints. The constraints specify a certain relation over a subset of variables. The relations limit the values that the variable can have. Different constraints engage different variables making a network of constraints. The problem that requires to be solved is finding a relation over the entire network of variables that simultaneously satisfies all constraints. The derived problem type is named Constraint Satisfaction Problem – CSP. This methodology is perfectly suited for schedule generation, since the entities engaged can be defined and the expected correct schedule can be declaratively expressed. If the object-oriented approach is added, the result will be general, in the same time having the possibilities for exact specialization.
Constraint programming is a term close to mathematical programming (Sedgewick, 1983). Mathematical programmes contain a set of variables interconnected by a set of mathematical equations called constraints and an objective function that calculates the quality of the solution represented by certain combination of values for the variables. If all equations are

only linear combinations of variables, the problem is a special case named linear programming.
After the problem is modelled with constraints, the state space derived from the variable domain and the constraints requires to be searched for the best solution. The algorithms for searching the state space are a key phase in the solving process.
Constraint Optimization Problems (COP), also known as Constraint Relaxation Problem – CRP (Yoshikawa, 1996), can be defined as common problems of constraint satisfaction in which the level of satisfaction of every constraint can be measured. The goal is to find a solution that maximises the sum of constraint satisfactions. Also, constraint optimization problems can be defined as constraint satisfaction problems upgraded with several local cost functions. The goal of the optimization is finding a solution whose cost, evaluated as a sum of all cost functions, should be maximal or minimal. Regular constraints are called hard, while cost functions are known as soft constraints. Names illustrate that hard constraints must be satisfied, while the soft ones only express preferability toward some solutions.

3. Metaheuristic algorithms and their hybridization

In recent decades we have witnessed the development of a new kind of approximative algorithms that combine basic heuristic methods in frameworks designed for efficient and effective search of the state space. These methods are named metaheuristics. The term was suggested by Glover in 1986, based on the ancient words: "heuristic" meaning "to discover", and the prefix "meta" meaning "above, higher level". These groups include, among others: Ant Colony Optimization (ACO), Evolutionary Computation (EC) – like Genetic Algorithms (GA), Iterated Local Search (ILS), Simulated Annealing (SA), Tabu Search (TS), Brute-force search, Random optimization, Local search, Greedy algorithm, hill-climbing, Random-restart hill climbing, Greedy best-first search, Branch and bound, Swarm intelligence - Ant colony optimization, Greedy Randomized Adaptive Search Procedure – GRASP etc. There is no strict definition for what metaheuristic is, but main axioms found in literature state that metaheuristics are a group of strategies that guide the search process. They search for an optimal or near optimal solution aproximatively and non-deterministically (Blum & Roli 2003).
The combination of different heuristic can be done in several ways. Various heuristic methods can be chronologically applied in different phases of the search, when their advantages are required the most. Besides chronological sequential application of different search methods, the algorithms themselves can be a hybrid of more metaheuristic or basic optimization approaches.
There are various forms of hybridization of algorithms. The first form advocates integration of components from one metaheuristic into another. The second form includes systems known as cooperative search. They are consisted of various algorithms that exchange information. The third option is integration of approximative and systematic (complete) methods. By emphasizing the advantages and flaws of different metaheuristic approaches, it is evident that hybridization and integration of different heuristic algorithms might result in better solutions to problems.
Since schedule generation is a NP hard problem, methods for exhaustive search are not an option. Algorithms like Tabu search, Genetic Algorithms and Simulated Annealing have been previously applied to such problems. Although they all have advantages, no one solves the problem completely. The goal of our research was to implement combinations of algorithms.

3.1 Exchange of components between metaheuristics

A popular way of hybridization is the use of trajectory methods with populations based methods (Blum et al., 2005). The most successful applications of evolutionary algorithms and ant-colony optimization use procedures for local search. The reasons are obvious when the appropriate advantages of trajectory and population methods are analysed.

The power of population methods is based on recombining solutions to derive new ones. Evolutionary algorithms and Scatter search implement explicit recombination with one or more operators (Glover et al., 2003). In Ant-colony optimization and some evolutions algorithms, the recombination is implicit, because new solutions are generated by using a distribution in the state-space, based on the previous populations. This allows guided steps in the search space that are usually bigger than steps made in trajectory methods. That means the solution based on recombination in population methods is more "different" from the parents as opposed to a solution derived with one move from the previous solution. There can also be "big steps" in trajectory methods, like iterated local search and variable neighbourhood search, but, in these methods the steps are not guided (these steps are called trials or perturbations to emphasise the lack of guidance). In every population based method, there are mechanisms that use the good solutions that have been found to influence the search to find even better solutions. The idea has been explicitly implemented in the Path Relinking algorithm (Blum et al., 2005). There, the basic elements are initial solutions and guiding solutions (the best found so far). New solutions are derived by applying moves to decrease the distance between the resulting and the guiding solution. Evolution algorithms achieve the same effect by keeping the best found solutions so far in the population. The approach is called an evolution process with stable states. Scatter search performs a process with stable states. In some implementations of ant colony optimization, there is a schedule for updating the pheromones that uses only the best found solution when the algorithm converges toward the end. It corresponds with changing the direction of the search process toward a good solution hoping to find better on the way.

The power of trajectory methods is the way that they search the promising regions in the state space. Since local search is the main component, a promising part of the search space is searched in a more structural way than in population based methods. This approach reduces the possibility of missing the optimal solution when the search is near it, as opposed to population methods. The conclusion is that population methods are better in identifying promising regions in the search space, while trajectory methods are better in exploring the promising area. Therefore, metaheuristic methods that combine the advantages of population and trajectory methods are successful.

3.2 Cooperative search

A loose form of hybridization is achieved in joint search (Hogg & Huberman, 1993) which consists of searching by various algorithms that exchange information for states, models, entire sub problems, solutions or other specifics of the search space. Usually, the solving process is based on parallel execution of algorithms with different level of communication. The algorithms may be entirely different, or instances of the same algorithm functioning on different models or different configuration parameters. The algorithms that make the joint search can be aproximative or complete, or a mixture of aproximative and complete methods. Cooperative search receives increased interest because of the interest in parallelisation of metaheuristic.

3.3 Integration of metaheuristic and systematic methods

The approach of integration of metaheuristic and systematic methods is quite effective when used over practical problems. The discussion about similarities, differences and possible integration of metaheuristic and systematic methods can be found in (Glover & Laguna, 1997). Recent research papers suggest that integration of metaheuristic and constraint programming is proving especially useful and successful. (Duong & Lam, 2004), (Gomes et al., 2005), (Crawford et al., 2007)

3.4 Hybridization with parallelization

Some of the heuristic algorithms are inherently easily executed in parallel, while others require sophisticated strategies for parallel execution. Generally, genetic algorithms (GA) are easy to execute in parallel, while SA is sequential by nature. On the other hand, there is a mathematical proof that SA slowly, but surely converges toward the final solution. Since there is no such proof for the GA, a hybrid SA with operators from genetic algorithms is a good approach.

There are various Parallel Genetic Simulated Annealing Algorithms – HGSA. In literature [Ohlidal 2004] there are a couple of published versions:

- S. W. Mahfoud and D. E. Goldberg suggest a concept of a GA using a Metropolis algorithm in the selection process.
- M. Krajic describes a hybrid parallel SA based on genetic operators (mutation and cross-reference).
- N. Mori, J. Yoshida and H. Kita use a thermodynamic rule for selection.

Czech describes a parallel implementation of SA without GA. (Czech et al., 2006)

3.4.1 Parallel SA with a Boltzmann synchronization function

Within our research we experimented with parallel execution of SA and developed parallel SA with a Boltzmann synchronization function. (Chorbev et al., 2006)

The cooperation of more processors can be used either to speed up the sequential annealing algorithm or to achieve a higher accuracy of solutions to a problem. In this work we considered both goals. The accuracy of a solution is meant as its proximity to the global optimum solution.

We designed a system with *r* available processors and each of them is capable of generating its own annealing process. The architecture includes a master computer and given number of slave computers, interconnected in a Local Area Network. The starting - master computer P_1 imports the initialization data, generates the first proposed solution and passes data to *r-1* remaining computers. All remaining computers – processors start independent annealing process after receiving the initial data. All processors communicate by exchanging current best solutions during annealing processes, at a chosen rate. The scheme of communication is given in the figure 1.

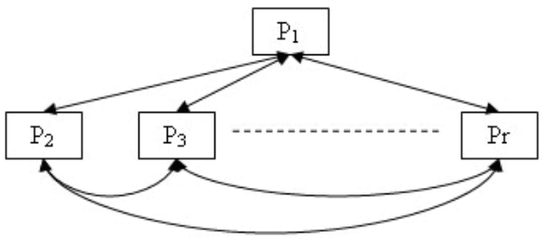


Fig. 1. Processor communication

The communication model used is synchronized point-to point. Before the temperature is decreased, every process sends its best found solution to the remaining *r-2* processes, and waits the best found solutions from all other processes, too. Once all data is received, each process calls its acceptance function to decide whether to accept the best solution from all other processes or continue with the one found by the process itself. With this architecture, the master computer only starts the solving process and eventually, collects best solutions from slave computers. It serves no purpose during solving iterations and information exchange; therefore its functions could be performed by some of the slaves. Keeping master's functions limited excludes it being a bottleneck in the architecture.
We analyzed a possible problem of certain faster converging processes to wait for slower processes to send their best solutions. This architecture is only as fast as the slowest of the included computers. However, we consider this not to be a setback. All computers used in the network are of same type, design and performance. Also, all computers execute the same annealing algorithm; use the same temperature decrement coefficient, the same number of iterations during each temperature and the same metropolis function. The only difference is the independent random generation of the next proposed solution in every computer. This provides different search paths through the solution space in every parallel process and increase diversity of the search. Therefore, all computers are expected to make at average the same number of acceptance and declination of new proposed solutions (due to the metropolis function). The cumulative result is roughly the same computational effort (time of execution) in each computer. Sometimes some processors might converge faster toward a local optimum, but the necessary broad search of the domain that this parallel architecture brings is worth waiting.
The acceptance function (the decision in every processor to accept the best solution from others or continue with its own) was also a subject of interest in our research. We tried: always accepting the best solution from all others, randomly accepting any of the given solutions from other processes and eventually accepting solutions using the Boltzmann distribution. We got the best results using the Boltzmann distribution. This probability function is fundamental for SA and it seems natural for it to be part of SA's parallelization.
There are other points among the algorithm steps where parallel processes could communicate, i.e. different rates of communication. Data could be exchanged within the inner annealing iteration at every n^{th} iteration or after certain number of temperature decreasing iterations. In our parallel SA the processes P_2, P_3, …, P_r cooperate among each other at every temperature decreasing iteration.
Implementation of the parallel SA is the following:

Process P_0:
INITIALIZE;
Dispatch initial solution to processes Pp, p=2, 3, …, r
Wait until final solutions from processes Pp, p=2, 3,…, r are received
Choose and display the best solution from processes Pp, p=2, 3,…, r
Process Pp, p=2, 3,…, r:
INITIALIZE;//receive initially proposed solution
repeat
    repeat
        PERTUB(solution(i) -> solution(j), Δcostij);
        if METROPOLIS(Δcostij) **then** accept

if accept **then** UPDATE(solution(j));
until predefined number of iterations;
Send current solution s(p) to other processes Pq, q=2, 3,…, r, q≠p
Receive solutions from all proc. Pq, q=2, 3,…, r, q≠p
Choose the best solution s(q) from received solutions
if $\exp(-\Delta cost_{pq}/TEMP) > random[0; 1)$ //Boltzmann
then accept;
if accept then UPDATE (solution s(j));
$TEMP_{+1} = f(TEMP)$; //Decrease temperature
until stop criterion = true (system is frozen);

A crucial component when designing a parallel algorithm is finding the best tradeoff between the amount of communication and every processor's independence. Communication of the parallel processes within the inner annealing cycle causes extensive communication slowing the overall performance. On the other hand, delaying the communication for every n^{th} temperature iteration gave worse solution quality because of lack of sufficient information exchange. The experimental results given in figure 2 show that best results are attained when communicating at every temperature iteration. This graph is generated with 5, 10 and 20 processors.

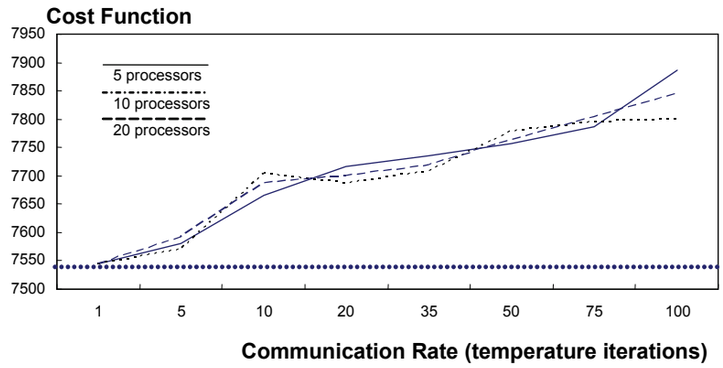


Fig. 2. Course of solution quality versus the rate of communication among processes. The horizontal axis is the number of temperature iterations between the processes communication. The vertical axis is the solution cost. The dashed line is the optimal solution

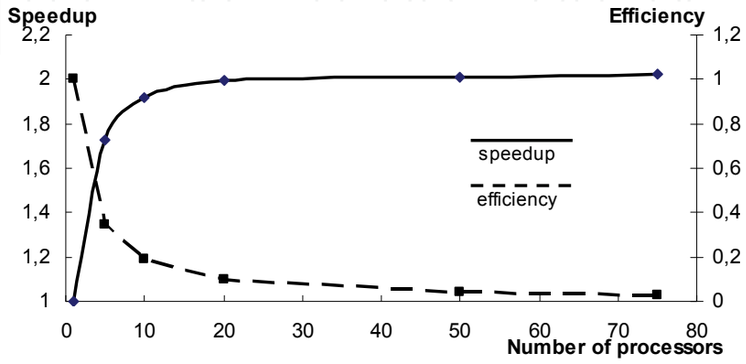


Fig. 3. Course of speedup and efficiency versus the number of processors.

Besides increasing solution quality, main reason for parallelization is the expected speedup. Speedup is defined as the ratio of solving time using single processor versus multiple parallel processors solving time. Efficiency is defined as the ratio of speedup versus the number of processors used. Efficiency gives the utilization of the processors. According to experimental results in figure 3, the speedup is obviously increasing when going from one processor toward five or ten. Further increasing of the number of processors brings no advantage since large amount of communication among processes slows the overall performance. According to experimental results, our parallel SA implementation achieves best speedup at 10 – 20 processors. However, if we take the efficiency into consideration, using more than 10 processors is highly inefficient.

4. Constraint solving engine with multiple optimization algorithms

The research presented in this chapter is performed using a Constraint Programming Library (CPL) (Jolevski et al. 2005a). The software library is consisted of a set of classes – generic constraint types for modeling different problem types and a mechanism for selection an optimal algorithm for the given problem. This approach is required because the Constraint Solving Engine (CSE) is developed to enable solution of problems with different nature. The engine is modular, allowing specific heuristics for certain problems to be implemented in overridden functions. Every step of the problem solving process could be implemented either with existing components or with newly added modules overriding those already contained in the basic object-oriented system.

The CSE is based on the concept of variables and their domains. The domains are bounded by the existing constraints in the moment of their creation, making the search space smaller. Later in the process of proposing new solutions, the constraints evaluate the extent of satisfaction and measure the progress toward the best (final) solution.

The main constraint class provides an integrated interface to all its children. The inherited interface enables algorithms to use the constraints, gives them their variables and checks the consistency of conditions. The constraints return Boolean or in some cases a quantitative measure of the constraint satisfaction.

In our model, the solution cost originates from the level of satisfaction of every constraint. Every constraint has an implemented function for calculation of the amount of its dissatisfaction. Additional multipliers to the dissatisfaction levels exist, to increase the influence of certain constraint over others in the total cost.

The set of given constraints is appropriate for modeling different problems. That is so because most of the problems can be divided into smaller and simpler ones that later can be modeled and solved. From mathematical point of view a broad variety of common, appearing different from the outside, problems are turned into the same or similar tasks.

When modeling a problem it can always be expressed through a mathematical language. Very often, the problem can be expressed as a couple of arrays of integer values that comply with certain rules. For the user, those model arrays are converted into understandable solution data like the shortest path for a traveling salesman or into the most optimal high school schedule. In the background, in the mathematical model, the problem rules transform into constraints like: "no two elements of the array can have the same value" or "the sum of all elements of the array must always be a constant value." All rules and value checks are done by methods within the constraints classes.

5. Constraints modelling of the high school scheduling problem

Scheduling covers a wide area of problems with temporal and spatial distribution of resources. Three broad families of scheduling problems can be distinguished depending on the degrees of freedom in positioning resource supply and resource demand intervals in time (Abramson, 1991): pure scheduling problems, pure resource allocation and finally, joint scheduling and resource allocation problems. High School scheduling is a composite problem.

In case of School Scheduling, the model includes means for temporal and spatial distribution of resources. It is required to implement priorities among constraints, providing methods for satisfying primarily more important rules followed by less significant.

The main interest of constraint programming lies in actively using the constraints to reduce the computational effort required to solve a problem, in the same time achieving good declarative problem formulation. Constraints are used not only to test the validity of a solution, but also in a constructive mode to deduce new constraints and detect inconsistencies. This process is called constraint propagation.

This problem domain falls within the category of Constraint Optimization Problems (COP), where the constraint(s) satisfaction requires to be evaluated (in opposition to those problems where they can only be satisfied or unsatisfied, called constraint satisfaction problems, CSP) (Penya et al., 2005). Application of algorithms like Simulated Annealing (SA) demands a solution cost function that the algorithm will tend to decrease (Leenen et al., 2003). Therefore, the implemented constraints of the model are capable of producing a numerical measurement of their satisfaction. Abramson (Abramson, 1991) in his model separates the total cost to three parts: teacher cost, class and room cost as a result of clashes in the trial solutions on those three bases.

Schedule generation has been formalized as a problem of optimizing constraints, or Constraint Relaxation Problem – CRP by (Yoshikawa et al., 1996). They focused on using the min-conflict heuristics to generate an initial solution for solving both school and university timetabling problems. After a fairly good-quality initial solution is generated by an Arc-Consistency algorithm, their proposal relies on a heuristic billiard-move operator to iteratively repair the current solution and complete assignment of lessons for school/university timetabling. The min-conflicts heuristic (MCH) tries to examine each variable to assign a value with the minimum number of constraint violations. Tam and Ting (Tam & Ting, 2003) combine the min-conflicts and look-forward heuristics used in local search methods to effectively solve general university timetabling problems.

5.1 Notation

The problem in our case is modeled as follows: There are G·D·N (G – Groups, D – Days, N – lessons per day) variables (items) that define the assignment of lessons to groups and rooms. Variables are grouped in blocks of D·N variables. Every block corresponds to the timetable for one group.

Let's denote the set of all lessons in a timetable by $T = \{t_0, t_1, \ldots, t_{T-1}\}$, where $T = |T| = G \cdot D \cdot N$ is the number of lessons in a timetable. Lessons are grouped in blocks of N lessons that are all in the same day. Lessons are ordered in an increasing day order.

The next stage, before actually turning the constraints into program code, is creating the mathematical model. For that purpose, an exact notation was required, part of which is defined as follows:

$\mathbf{\Pi} = \{\pi_0, \pi_1, \ldots, \pi_{P-1}\}$ set of teachers;
$\mathbf{B} = \{\beta_0, \beta_1, \ldots, \beta_{B-1}\}$ set of subjects;
$\mathbf{\Psi} = \{\psi_0, \psi_1, \ldots, \psi_{Y-1}\}$ set of school years;
$\mathbf{\Delta} = \{\delta_0=0, \delta_1=1, \ldots, \delta_D=D\text{-}1\}$ set of working days;
$\mathbf{\Gamma} = \{\gamma_0, \gamma_1, \ldots, \gamma_{G-1}\}$ set of groups;
$\mathbf{X} = \{\chi_0, \chi_1, \ldots, \chi_{C-1}\}$ set of rooms;
$\mathbf{I} = \{I_{min}, I_{min}+1, \ldots, N\}$ set of number of lessons in a day;
I_{min} minimum number of lessons in a day; and
N maximum number of lessons in a day.

5.2 Data

The algorithm works using existing data. The data has to be previously entered in the program (in the relational database through an intuitive user interface of the scheduling software (Jolevski et al., 2005d)), and prepared in the specified format. Certain data is exploitive in more than one constraint. Our model contains eight previously entered data structures. They are:

1. Number of weekly lessons x per subject β per group γ is defined by the following set of ordered triples: $\mathbf{B\Gamma} = \{(\beta_i, \gamma_j, x) \mid i = 0, \ldots, B\text{-}1; j = 0, \ldots, G\text{-}1\}$.
   Function $x = X_{\beta\gamma}(\beta, \gamma)$: $\mathbf{B\times\Gamma} \rightarrow \mathbf{Z}_+$ returns the required number of lessons for subject β for group γ.
   Function $x = w_\gamma(\gamma)$: $\mathbf{\Gamma} \rightarrow \mathbf{Z}_+$ returns the required number of weekly lessons for group γ.
2. All combinations of groups and subjects that a particular teacher can teach are given in the following set. Teacher π who teaches subject β for group γ is defined by the following set of ordered triples: $\mathbf{\Pi B\Gamma} = \{(\pi_k, \beta_i, \gamma_l) = \mid k = 0, \ldots, P\text{-}1; i = 0, \ldots, B\text{-}1; l = 0, \ldots, G\text{-}1\}$.
   Function $\pi = P_{\beta\gamma}(\beta, \gamma)$: $\mathbf{B\times\Gamma} \rightarrow \mathbf{\Pi}$ returns the teacher π for subject β for group γ.
3. Subjects β that can be taught in a room χ is defined by the following set of ordered pairs: $\mathbf{BX} = \{(\beta_i, \chi_j) \mid i = 0, \ldots, B\text{-}1; j = 0, \ldots, C\text{-}1\}$.
   Function $T_{\beta\chi}(\beta, \chi)$: $\mathbf{B\times X} \rightarrow \{0, 1\}$ returns 1 if subject β can be taught in room χ, otherwise returns 0.
4. Maximum number m of groups that can share a room χ is defined by the following set of ordered pairs:
   $\mathbf{X}_M = \{(\chi_i, m) \mid i = 0, \ldots, C\text{-}1\}$.
   Function $M_\chi(\chi)$: $\mathbf{X} \rightarrow \mathbf{Z}_+$ returns the maximum number of groups that can share room χ at any point in time.
5. If a subject β should be taught in blocks of 2 consecutive lessons, than there is an appropriate element in the set $\boldsymbol{C} = \{\beta \mid \beta \in \boldsymbol{B}\}$
   The function $c_\beta(\beta)$: $\boldsymbol{B} \rightarrow \{0,1\}$ returns 1 if the subject β can be taught in blocks of 2 lessons, because $\beta \in \boldsymbol{C}$, else returns 0 meaning $\beta \notin \boldsymbol{C}$.
6. The availability of the teacher in a particular time slot during the week is kept in this set. A teacher π_i available to teach a class a on a day δ_j in a shift μ_k is defined with the following set:

$$\boldsymbol{A} = \{(\pi_i, \delta_j, \mu_k, a) \mid i = 0, \ldots, P - 1; j = 0, \ldots, D - 1; k = 0, \ldots, M - 1\}$$

If elements exist for a professor in the set $\boldsymbol{\Pi\Delta M}$, then these elements hold the availability of the teacher. If there is no element for the particular teacher in the set $\boldsymbol{\Pi\Delta M}$ the teacher is available at any time in the week.

The function $a_\alpha(\pi_i, \delta_j, \mu_k, a)$: $\boldsymbol{A} \rightarrow \{0,1\}$ returns 1 if the teacher π_i is available to teach the subject a in the day δ_j in shift μ_k , that is $(\pi_i, \delta_j, \mu_k, a) \in \boldsymbol{A}$, else returns 0 meaning $(\pi_i, \delta_j, \mu_k, a) \notin \boldsymbol{BA}$.

7. The subject β can not be taught in a lesson j:

$$\boldsymbol{BA} = \{(\beta, j) \mid \beta \in \boldsymbol{B}; j \in \{1, \ldots, N\}\}.$$

The function $B_\alpha(\beta, j)$: $\boldsymbol{B} \rightarrow \{0,1\}$ returns 1 if the subject β can be taught in the lesson j, meaning $(\beta, j) \notin \boldsymbol{BA}$, else returns 0 meaning $(\beta, j) \in \boldsymbol{BA}$.

8. The set of elective subjects is $\boldsymbol{E} = \{\beta \mid \beta \in \boldsymbol{B}\}$.
The function $e_\beta(\beta)$: $\boldsymbol{B} \rightarrow \{0,1\}$ returns 1 if β is elective subject or $\beta \in \boldsymbol{E}$, else returns 0 meaning $\beta \notin \boldsymbol{E}$.

5.3 Constraints

The problem is modeled by representing it through 16 constraints. They are defined as follows:

1. $\sum_{k=i}^{i+D\cdot N-1} (b_\tau(\tau_k) = \beta_j) = X_{\beta\gamma}(\beta_j, \gamma_i)$ for $i = 0, D\cdot N, 2\cdot D\cdot N, \ldots, (G\text{-}1)\cdot D\cdot N$ and $j = 1, 2, \ldots, B$, where the sum represents the number of weekly lessons for subject β_j in group γ_i. The number of weekly lessons per subject in a group is defined by a set in the entered data.
When this constraint was coded with the given tools in the Constraint Solving Engine, in the implementation, the generic constraint class CSetCover was used. The generic constraint classes are developed universally so that they could be used in different forms to express different real problem constraints. The CSetCover constraint class, as all other in the CSL, inherits from the basic constraint class (Chorbev et al., 2007). Its task is to check the number of occurrences of a certain value for a given variable coordinate in the array of variables (the "subject" value of the complex time slot variable in this case). When the number of occurrences of the given value in the variable is adequate, the constraint is "covered". The set of values to be covered was equal to the set of courses **B**. Every set value requires to be covered exactly $X_{\beta\gamma}(\beta,\gamma)$ times. Since the set to be covered can be different for different groups, a separate instance of the CSetCover class is necessary to be created for every group.
2. $\sum_{k=i}^{i+D-1} (l_\nu(\nu_k) - f_\nu(\nu_k) + 1) = w_\gamma(\gamma_m)$ for $i = 0, D, 2\cdot D, \ldots, (G\text{-}1)\cdot D$, m = i/D, where the sum is the number of weekly lessons for group γ_m. Number of weekly lessons per group is defined by a set in the entered data.
3. $T_{\beta\chi}(b_\tau(\tau_i), r_\tau(\tau_i)) = 1$ for $i = 0, 1, 2, \ldots, G\cdot D\cdot N\text{-}1$
Subjects are always taught in appropriate rooms as defined by the input data.
4. $a_\tau(\tau_j) = 0$ for $i \le j < i+f$ and $i+l+1 < j \le i+N\text{-}1$ for $k = 0, 1, \ldots, G\text{-}1$ and $l = 0, 1, \ldots, D\text{-}1$, $i = k\cdot D\cdot N + l\cdot N$, where $n = k\cdot D+l$, $f = f_\nu(\nu_n)$, $l = l_\nu(\nu_n)$. There can be no timetable breaks. Empty lessons are determined by variables ν_n.
5. $a_\tau(\tau_j) = 1$ for $i+f \le j \le i+l$ for for $k = 0, 1, \ldots, G\text{-}1$ and $l = 0, 1, \ldots, D\text{-}1$, $i = k\cdot D\cdot N + l\cdot N$, where $n = k\cdot D+l$, $f = f_\nu(\nu_n)$, $l = l_\nu(\nu_n)$. There can be no timetable breaks. Non-empty lessons are determined by variables ν_n.

6. $\sum_{i=1}^{G} \begin{cases} 0 & if \quad r_\tau(\tau_{(i-1)\cdot D\cdot N+l}) \neq \chi_j \\ 1 & if \quad r_\tau(\tau_{(i-1)\cdot D\cdot N+l}) = \chi_j \end{cases} \leq M_\chi(\chi_j)$ for $j = 1, 2, \ldots, C$ and $l = 1, 2, 3, \ldots, D\cdot N$

   At any point in time l, room χ_j can have at most $M_\chi(\chi_j)$ groups as defined by the set $\mathbf{X}_M$ in the input entered data.

   As it was described in the previously published material (Chorbev et al., 2007, Jolevski et al., 2005b) the constraints classes implemented in the Constraint Solving Library (CSL) were created to be universal and applicable to different problems. Therefore, in this constraint the class CSetCover (generic constraint) was used. The set of values to be covered was $\mathbf{X}_M$. In the constraint-variable network, practically the model of the problem, there were $D\cdot N$ copies of this constraint. There was a copy for each time slot (lesson) in the work week. However, every copy is in fact the same instance of the generic CSetCover constraint, since the set of values to be covered was always $\mathbf{X}_M$. In every different copy, the classroom coordinate of the variables for different groups for that lesson was taken in consideration.

7. $b_\tau(\tau_i) \in \mathbf{A} \wedge i \neq l_\nu(\nu_k) \wedge (b_\tau(\tau_i) \neq b_\tau(\tau_{i-1}) \vee i = f_\nu(\nu_k)) \Rightarrow b_\tau(\tau_{i+1}) = b_\tau(\tau_i)$ for $i = 0, 1, \ldots, G\cdot D\cdot N-1$, and $k = i$ div N.

   If subject $b_\tau(\tau_i)$ must be taught in blocks of 2 lessons, and τ_i is not the last lesson in the day k, and subject $b_\tau(\tau_i)$ is different from the previous subject $b_\tau(\tau_{i-1})$ or τ_i is the first lesson in a day, then the next lesson needs to be for the same subject $b_\tau(\tau_{i+1})$.

8. $b_\tau(\tau_i) \neq b_\tau(\tau_j)$ for $k = 0, 1, \ldots, G-1$ and $l = 0, 1, \ldots, D-2$, where $i = l_\nu(\nu_n)$ and $j = f_\nu(\nu_{n+1})$, $n = k\cdot D+l$. n is the index of the variable ν_n that defines the index i for the last lesson τ_i in day l for group γ_k.

   Subject $b_\tau(\tau_i)$ for the last lesson in a day except for the last day in the week (usually Friday) and subject $b_\tau(\tau_j)$ for the first lesson in the previous day for any group must be different.

9. $b_\tau(\tau_{i+f}) \neq b_\tau(\tau_{i+f+1}) \neq \ldots \neq b_\tau(\tau_{i+l})$ for all $b_\tau(\tau) \notin \mathbf{A}$, for $k = 0, 1, \ldots, G-1$ and $l = 0, 1, \ldots, D-1$, $i = k\cdot D\cdot N + l\cdot N$, where $n = k\cdot D+l$, $f = f_\nu(\nu_n)$, $l = l_\nu(\nu_n)$. n is the index of the variable ν_n that defines the indices for the first and last lessons τ_{i+l} and τ_{i+l} in day l for group γ_k.

   Subjects for all lessons τ_{i+f} to τ_{i+l} in a day l must be different for all subjects that can not be taught in blocks of two lessons $b_\tau(\tau) \notin \mathbf{A}$.

10. $a_\alpha(p_\tau(\tau_i), j, m_\lambda(\lambda_l), k) = 1$ for $l = 0, 1, \ldots, G-1$; $j = 0, 1, 2, \ldots, D-1$; $k = 0, 1, 2, \ldots, N-1$; where $i = l\cdot D\cdot N + j\cdot N + k$.

    For all groups ($l = 0, 1, \ldots, G-1$), all days ($j = 0, 1, \ldots, G\cdot D-1$), and all lessons ($k = 0, 1, 2, \ldots, N-1$), the teacher $p_\tau(\tau_i)$ must be available for the lesson τ_i.

11. $B_\alpha(b_\tau(\tau_{i\cdot N+j}), j) = 1$ for $i = 0, 1, \ldots, G\cdot D-1$; $j = 1, 2, \ldots, N$.

    For all groups and all days ($i = 0, 1, \ldots, G\cdot D-1$), the subject $b_\tau(\tau_{i\cdot N+j})$ taught at any lesson ($j = 1, 2, \ldots, N$) must be allowed by the set **BA**.

12. $e_\beta(b_\tau(\tau_{f\cdot D\cdot N + j\cdot N+k})) = e_\beta(b_\tau(\tau_{(f+1)\cdot D\cdot N + j\cdot N+k})) = \ldots = e_\beta(b_\tau(\tau_{l\cdot D\cdot N + j\cdot N+k}))$ for $i = 0, 1, \ldots, Y-1$; $j = 0, 1, \ldots, D-1$; $k = 0, 1, 2, \ldots, N-1$; where $f = f_\psi(\psi_i)$ and $l = l_\psi(\psi_i)$.

    For all school years ($i = 0, 1, \ldots, Y-1$) and all days ($j = 0, 1, \ldots, D-1$), and all lessons in a day ($k = 0, 1, 2, \ldots, N-1$), the elective/non-elective property of the subject is equal for all groups.

13. $y_\gamma(\gamma_i) = y_\gamma(\gamma_j) \Rightarrow s_\lambda(\gamma_i) = s_\lambda(\gamma_i)$ for $i = 0, 1, \ldots, G-1$; $j = 0, 1, \ldots, G-1$; $i \neq j$

    If two groups γ_i and γ_j are in the same school year $y_\gamma(\gamma_i) = y_\gamma(\gamma_j)$, then they are in the same shift $s_\lambda(\gamma_i) = s_\lambda(\gamma_i)$.

14. $r_\tau(\tau_{i+k}) = r_\tau(\tau_{j+k}) \wedge s_\lambda(\gamma_m) = s_\lambda(\gamma_n) \Rightarrow b_\tau(\tau_{i+k}) = b_\tau(\tau_{j+k})$ for $i = 0, D{\cdot}N, 2{\cdot}D{\cdot}N,\ldots, (G{\cdot}D\text{-}1){\cdot}N$; $j = 0, D{\cdot}N, 2{\cdot}D{\cdot}N,\ldots, (G{\cdot}D\text{-}1){\cdot}N$; $i \neq j$; $m = i/(D{\cdot}N)$; $n = j/(D{\cdot}N)$; $k = 0, 1, \ldots, D{\cdot}N\text{-}1$.
If two groups γ_m and γ_n from the same shift $s_\lambda(\gamma_m) = s_\lambda(\gamma_n)$ are scheduled to share room $r_\tau(\tau_{i+k}) = r_\tau(\tau_{j+k})$ during lesson k, then lesson's subject will be same for both groups.
15. $r_\tau(\tau_{i+k}) \neq r_\tau(\tau_{j+k}) \wedge s_\lambda(\gamma_m) = s_\lambda(\gamma_n) \Rightarrow p_\tau(\tau_{i+k}) \neq p_\tau(\tau_{j+k})$ for $i = 0, D{\cdot}N, 2{\cdot}D{\cdot}N,\ldots, (G{\cdot}D\text{-}1){\cdot}N$; $j = 0, D{\cdot}N, 2{\cdot}D{\cdot}N,\ldots, (G{\cdot}D\text{-}1){\cdot}N$; $i \neq j$; $m = i/(D{\cdot}N)$; $n = j/(D{\cdot}N)$; $k = 0, 1, \ldots, D{\cdot}N\text{-}1$.
If two lessons τ_{i+k} and τ_{j+k} that happen at the same time k and in the same shift $s_\lambda(\gamma_m) = s_\lambda(\gamma_n)$ are held in different rooms $r_\tau(\tau_{i+k}) \neq r_\tau(\tau_{j+k})$, the teachers must be different $p_\tau(\tau_{i+k}) \neq p_\tau(\tau_{j+k})$.
16. Timetable breaks for teachers are minimized.

6. Implementation of a hybrid simulated annealing algorithm

When solving the school scheduling problem, we attempted to add additional functionalities from other optimization algorithms in Simulated Annealing (SA). We started by SA knowing of its power to avoid local optima and its theoretical guaranty to find the global optimum. SA has been extensively researched and has shown satisfactory results in solving problems of combinatorial optimization and temporal and spatial scheduling (Duong & Lam, 2004), (Abramson, 1991), (Aarts et al., 2003), (Czech et al., 2006). In the software library, we implemented a combined version of the SA algorithm. It has elements of memory from the Tabu Search as well as a complex neighborhood function for local search similar to the Guided Search algorithm.
Detailed explanation of the algorithm follows after the pseudo code:

```
initialSol ← ConstructAsCorrectInitSolutAsPossible();
SolutionNeighborhood.SetSolution( initialSol );
{listOfVarsToChange, currentEnergy} ← CalculateEnergy(initialSol);
temp = InitialTemperature;
do
do
SolutionNeighborhood.GenerateCandidateSol ( currentSol, listOfVarsToChange);
FindAffectedConstraints();
{listOfVarsToChange,newEnergy}  ←  CalcEnergy(SolNeighborhood.Candidate);

if  ( Metropoliten(newEnergy - currentEnergy,temp))
                SolutionNeighborhood.AcceptCandidate();
else
                SolutionNeighborhood.RefuseCandidate();
endif

while ( !stopSearch and (trials < saMaxTrials )
        and successfulTrials < saMaxSuccessfulTrials
        and smallestEnergy > lowerEnergyBound );

temp = TempSchedule.GetNewTemperature();

while ( !stopSearch
```

and numberOfTempDecreases < MaxTempDecreases
and SolutionCount < MaxNumberOfSolutionsToFind
and smallestEnergy > lowerEnergyBound
and consecutiveNoSuccess < MaxConsecNoSuccess **)**

return SolutionNeighborhood.ReturnBestFoundSolution();

The algorithm initially constructs the solution in a way that as many as possible constraints are satisfied, and the cost - energy is reduced to minimum. In the particular high school schedule generation, for every group, in the available time slots, the appropriate number of classes per subject is filled. A teacher is assigned for every subject. All lessons are inserted in the time slots continually, until the appropriate numbers of classes per subject per group are achieved. The remaining task for the algorithm is to move the items (group, subject, teacher) in other time slots during the week, so that the remaining constraints are satisfied (not repetition of the same subject twice in the same day etc.)
In the meantime, an object from the CNeighborhood class is generated (Jolevski et al., 2005a). This object holds the current solution, and when asked for, generates a new proposal solution in the neighborhood of the current one. This is the place where the local search in the search space is performed.
After the neighborhood function generates a new solution, the algorithm invokes the function FindAffectedConstraints(). It detects the constraints whose satisfaction has been changed during the previous solution perturbation. Having this information, the function CalculateEnergy() calculates only the participation of the affected constraints within the overall energy, as opposed to recalculating the entire cost. Additionally, the function CalculateEnergy() generates a new list listOfVarsToChange. The list states which variables to change in the next solution perturbation so that a better solution is derived.
The Metropoliten() function implements the Metropolis probability distribution function. Considering the difference of energies of the previous and actual solution, as well as the temperature parameter, it decides whether to accept or reject the new solution.

$$P_T(\Delta E(X)) = \begin{cases} 1, & \Delta E(X) <= 0 \\ \exp(-\frac{\Delta E(X)}{T}), & otherwise \end{cases}$$

The inner iterations continue executing at constant temperature until one of the given conditions is met. Execution stops when the predefined maximal number of iterations per temperature is achieved, the maximally allowed number of accepted solutions is achieved, or the expected minimal energy is evaluated.
After ending the internal iterations, the function myTempSchedule.GetNewTemperature() is invoked. Depending on the chosen temperature schedule, a new value for the parameter temperature is derived. Having the new temperature, a new cycle of the internal iterations follows. If the conditions for ending the entire solving are met, the algorithm returns the best found solution. The ending conditions consist of achieving predefined number of temperature iterations, achieving minimal energy or achieving a predefined number of iterations where the metropolis function has not accepted any solution proposals.
In the presented implementation of SA we included functionalities from other metaheuristic algorithms. Memorizing and using the list of previously affected variables in the new

solution proposal in our algorithm includes memory in the solving. Memory is an element from Tabu Search. The list of affected variables helps in guiding the search and avoiding cycles. Generating a new solution proposal based on knowledge of the problem and previous experience adds elements of Guided search.

6.1 Neighborhood function for generation of an improved solution

Despite the effort to build a universal solving engine, there are certain parts of the system that seriously benefit from specialization. We estimated that the best part to implement specialization, in respect to both modularity and performance, is the neighborhood function. A neighborhood generation function is required to generate a new solution similar to, or in the "neighborhood" of, the previous one. The new solution is expected to have a lower cost (energy), meaning that the level of constraint satisfaction is higher. Presumably, the next solution should keep the qualities of the former and hopefully correct its weaknesses. The new solution proposal sometimes might have worse qualities than the previous one, but it still might be accepted as base of future solutions. Such acceptance has to be allowed to escape local in the pursuit of the global optima.

There are numerous ways to implement new trial solution generation. We used several combined approaches. The first method that we used, mainly during the first iterations of the solving process, is random based variable permutation. The method at the beginning considers how big part of the current solution should be changed in each iteration. We experimented with numbers from two variables up to 10% of the overall variables. The number of improvements in the solution during algorithm iteration increased as we decreased the number of changed variables. This behavior implies that a small change in the new solution is easier to find wrong and undo, while massive changes in the solution might compensate both corrections and faults, to a minor collective change in overall cost. Therefore, changing a small number of variables in each iteration makes the cost function an objective measure of progress or regress.

The neighborhood function randomly chooses which variables to change, keeps an array of changed variables and randomly assigns new values to the chosen variables from their domain. The array of changed variables is later used to calculate the new cost as a difference with the former one. Other approaches found in literature (Abramson, 1991) consist of swapping values of two variables. Nevertheless, for the sake of speed, the tendency is to always achieve calculation of the new cost as a difference with the last one.

6.2 Intelligent generation of the initial solution

The first place to implement intelligence in the solving process is the initial solution of the search. Significant number of iterations is avoided if the initial solution is not generated randomly, but in respect to the imposed constraints. Naturally, not all constraints could be satisfied at the start, otherwise, no search would have been necessary. We decided to satisfy as many constraints as possible at the beginning. Later during the solving process, initially satisfied constraints are not even checked for consistency, sparing many calculations in the evaluation step. This behavior is only possible if the neighborhood function includes special precaution. The functions must not dissatisfy these initial rules while generation of the next solution.

For example, all groups are initially given the right number of classes per subject. Therefore, no additional checks are necessary for this constraint, assuming the neighborhood function

only swaps positions of those classes in the available time and space slots. The neighborhood function must not add or subtract new classes per subject per group. Also, the classes are initially placed in rooms that they can be taught in.

6.3 Guided generation of new trial solutions

Different algorithm hybridizations based on SA for solving constraint modeled non-linear problems have been previously proposed. An example algorithm is CSAGA by Wah et. al (Wah & Chen, 2001). They combine SA and Genetic Algorithms (GA). The participation of the GA is in the creation of new solution proposals. A new generation of solution proposals is generated with genetic operators in each SA iteration. Every proposal is evaluated by multiplying its Lagrange-multipliers. The best proposal is selected with a GA.

In our SA implementation, additional intelligence was implemented in the neighborhood function. In the generation of the next solution, the neighborhood function uses listOfVarsToChange, the list of variables that it should change. The list of variables is derived when checking the constraint satisfaction and the particular variables that participate in the unsatisfied constraints. During the evaluation of the previous solution, the algorithm remembers which constraint generated the most of the unwanted cost. Knowing the variables that participate in the given constraint, the algorithm knows which variables should be changed to correct the current solution. This is how intensification of the search is achieved. Diversification is achieved by occasionally invoking a more complex neighborhood function, in a way explained later in the text.

One could favor the idea of generating the new solution by forcing changes in the variables that make the most of the unwanted cost at that point. However, we face two setbacks: the increased intelligence in the neighborhood function will decelerate the iterations; and the danger of trapping in local minima is increased. Nevertheless, random generation not always succeeds to find the final correct solution and therefore guided search is required to be implemented. Guided local search has already proven effective in solving the scheduling problem (Tsang et al., 1999).

We implemented guided search that includes different algorithms of swapping variable values. Depending on the constraint that has been dissatisfied, adequate neighborhood sub function is invoked. The neighborhood function swaps values of variables either in informed manner, generating a better solution, or in random position. For instance, if an empty class is spotted in-between classes, this empty time slot has to be filled with a class, so the last class of that day for the group is placed in that position, pushing the empty classes at the end. If two identical classes are found next to each other, one of them has to go in a different day. Swapping is performed between this class and a class from the next day, separating identical classes in different days.

Figure 4 presents the invoking of particular neighborhood sub-functions during algorithm iterations. The horizontal axis represents the algorithm iterations, while the vertical axes contains all 16 constraints that model the problem. Clearly, some constraints that were satisfied in the construction of the initial solution never have their sub-functions invoked later. The remaining constraints (5, 6, 7, 8, 15) are invoked with variable frequencies.

Figure 5 shows the distribution of calls of the particular constraint's sub-functions during the solutions search. The horizontal axis represents the constraints invoked during solving, and the vertical axis gives the number of calls of the given constraint. The most invoked constraint is the 15th, the collision when one teacher is placed to teach in two classrooms in

the same time. Because of its frequent invocations, a rather simple algorithm for correction has been developed. Two lessons from two time slots of one of the groups in the collision are chosen and swapped. For example, one of the collided lessons and another random lesson from the same group exchange their time slots. It is important to emphasize the random component that forces diversification in the solution search.

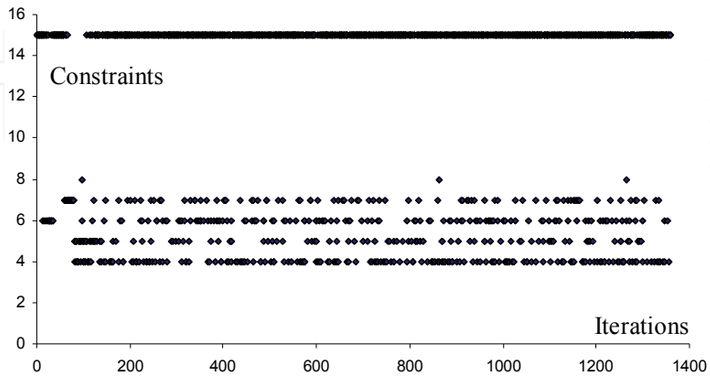


Fig. 4. Calls for evaluation of particular constraints of the model in every iteration of the solution search

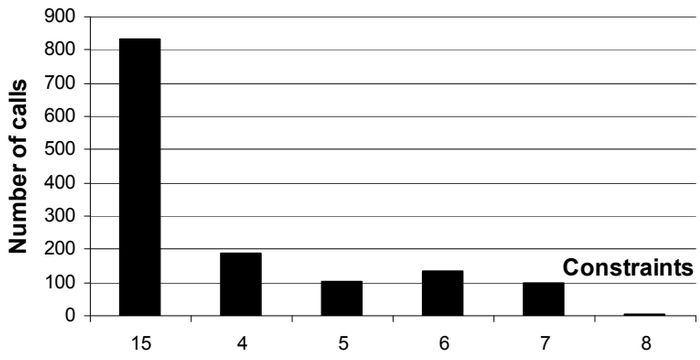


Fig. 5. Distribution of calls toward particular constraint's sub-functions during solving

6.4 Deadlocks

Practice has shown that simple variable swapping often brings to deadlocks. Initially a certain constraint is dissatisfied causing the appropriate swapping neighborhood function to swap values. The new solution dissatisfies a different constraint causing its swapping function to make the previous swap rollback. At this point a cyclic deadlock starts, with no possibility of ending. We solved this problem rather simply, but effectively. Every neighborhood function contains more than one swapping algorithm that triggers randomly, with different probability.

Certain more effective and easily calculable swapping methods are executed more often. Since they tend to cause deadlocks, once in a while, another different neighborhood function is triggered for the same constraint dissatisfaction (equation 1). For instance, if *rnd* is a randomly generated number such that $1 \leq rnd \leq 1000$, then:

$$NewSol = \begin{cases} SimpleSwap(OldSol), if\ 1 < rnd < 1000 \\ ComplexSwap(OldSol), if\ rnd = 1 \end{cases}$$

Varying the probability of triggering different neighborhood functions evidently influences solving time and result quality. Table 1 shows the dependences of solving speed and solution quality from the probability of using a more thorough permutation opposite to simple swap in the neighborhood function. A measure of solving quality is achieved minimal cost, number of made improvements, number of temperature decreases (SA parameter) and duration of the search.

Probabil.	Min. Cost	Number Improve.	Nr.Temp Decreases	Duration (msec)
1/5	315	68	2926	335857
1/10	355	80	5558	306890
1/100	326	86	3434	137983
1/1000	201	99	4273	133300
1/5000	297	99	4440	238560
1/10000	300	83	3214	259314

Table 1. Dependences of solving from the probability of using a complex neighborhood

Extremely high probabilities (0.2, 0.1) do not achieve the lowest solution cost, and solving duration is prolonged. Here the complex and thorough permutation is triggered too often and convergence toward the final solution is interrupted even when there is no deadlock. Extremely low probabilities (0.0001) also give unsatisfactory results because with such low probabilities, the thorough permutation is not triggered even when there is a deadlock going on. We chose to use a probability 1/1000 = 0.001, because it seems, the required neighborhood function triggers exactly when a deadlock happens.
Figure 6 shows the dependences of cost - energy in terms of SA, number of improvements and temperature decreases from the probability of using a complex neighborhood. It is visually evident that the chosen probability of 1/1000 = 0.001 gives the maximal number of improvements and the minimal energy - cost.

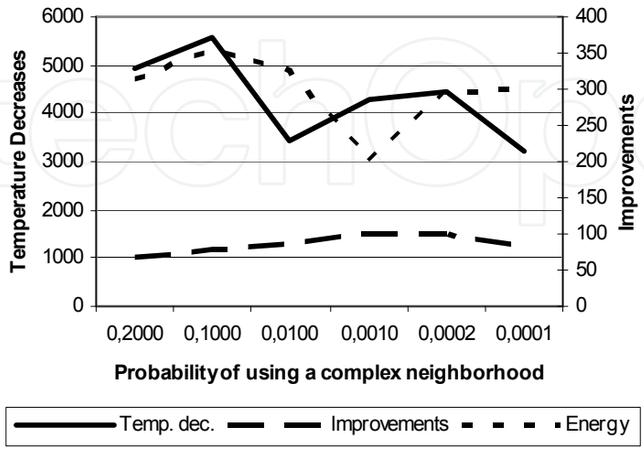


Fig. 6. Dependences of improvements, cost and energy from the probability of using a complex permutation.

Some of the parameters of SA must be functionally dependant from the input parameters of the problem. For instance, the number of iterations of the inner loop of SA or the number of overall temperature decrements of SA is a multiple of 100 and the number of variables in the problem model. The maximal number of algorithm executions without accepting new solutions is 100.

6.5 Impact of guided search

Our search through the solution space is guided in a manner such that the neighborhood function accepts the old solution proposal, along with the constraint that generates most of the cost function. Since the neighborhood function includes specific algorithms for solution improvement of each constraint, the appropriate one is triggered. The neighborhood function generates a new solution in the "neighborhood" of the current one, precisely marking the changed variables. Only the participation of the changed variables is recalculated into the overall solution cost. The recalculation function marks the constraint that contributes most to the cost.

Every constraint involves a different number of variables that create the problem model. Hence, every new solution generation changes a different number of variables depending on the constraint scope. We already determined (Chorbev et al., 2007) that changing a smaller amount of variables in every iteration gives better results. Still, letting the affected constraint decide the degree of change in the new iteration, allows appearing of occasional jumps in the search space. Massive change among the variables means jumping to new solution neighborhoods, when previous local search is exhausted.

The experimental results are generated from solving a school scheduling problem with 60 groups, 75 teachers, 37 rooms (utilized on two shifts) and 45 different subjects. Figure 7 shows the variations of the solution cost (energy) during the first 100 iterations of the solving process.

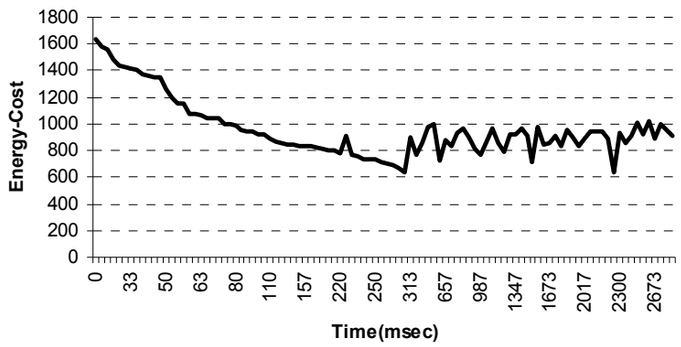


Fig. 7. Variation of solution cost in time during the first 100 iterations

The graph on figure 7 shows that informed - guided search in the beginning quickly directs the solution toward a lower solution cost. At that point the algorithm performs exact solving rather than heuristic search in each iteration. The neighborhood function exactly changes the dissatisfying variable for the most influential constraint to an accurate value. However, this process does not last long, quickly getting to a situation where every intervention among variables causes another equally influential constraint to become dissatisfied. In each succeeding iteration of the optimization algorithm, a triggered neighborhood corrects one, but dissatisfies other constraints.

This period of averagely constant cost, with a very small overall cost decrement lasts for most of the solving process (figure 8). During this period, swift jumps and depressions characterize the graph. The neighborhood function easily jumps to often completely new solutions. This behavior is inherited from Simulated Annealing, because high "temperature" allows the probability for acceptance of worse solutions to remain significantly big. As temperature decreases in SA and many of the neighborhoods have been tested, jumps start happening rarely. The search is directed in the neighborhood that has shown best potential for the final solution. The entire solving process can be seen at figure 8.

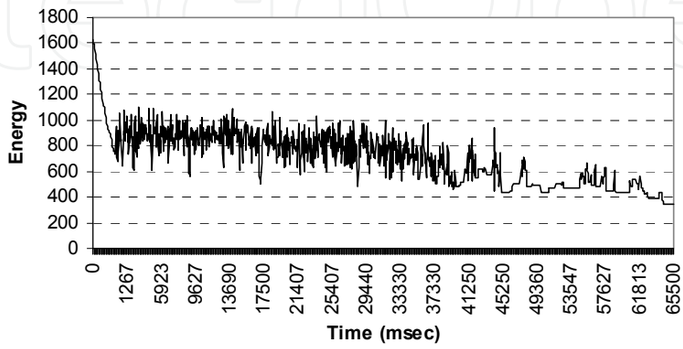


Fig. 8. Variation of solution cost in time during entire schedule generation

In this process, resulting from the nature of the algorithm SA, worsened solutions are accepted as bases for further improvement, therefore managing to escape the emerging deadlocks. Although the search is guided, and the functions change specific error variable values, randomness must still be present. The neighborhood functions swap values between time slots in the schedule, but often new positions for swapping values are given randomly. The randomness is controlled with the domain of the variables and the goal to satisfy other constraints. The combination of worse solution acceptance and randomness in new solution generation is the key to avoiding trapping in local optima and escaping deadlocks.

7. Conclusion

The chapter deals with description of the heuristic algorithm that we implemented to build a school scheduling software. The scheduling software is based on a Constraint Solving Engine (CSE) and a Constraint Programming Library (CPL) which we previously developed (Chorbev et al., 2007). Various simulations and tests of the solving process implied the required corrections to the model (Jolevski et al., 2005c) and the algorithms.

Every mentioned heuristic algorithm has certain advantages that come in handy in specific circumstances and specific problems. The goal in our research was to extract the best ideas and develop a novel hybrid algorithm that will achieve better performances. In this particular case, we tried to add additional functionalities from other optimization algorithms in Simulated Annealing as basis. We started from SA knowing of its power to avoid local optima and its theoretical guaranty to find the global optimum. We combined the memory from Tabu search, the intelligence of guided search and the completeness of GAC-CBJ. Initially we tested and fine tuned the algorithms on trivial problems like the traveling salesman, quadratic assignment or the n-queens problem. Eventually we took the schedule generation problem as a way to give practical implementation of the developed hybrid algorithms.

The area of constraint programming is quite perspective in the sense that it can use a lot of knowledge previously gathered from analysis in logical programming (Prolog). Furthermore, new better heuristic algorithms are implemented for solving problems modeled with constraints turning the constraint satisfaction into a perspective strategy. Even in worse case scenario, if solving through constraints does not give the promised results, the universal mathematical modeling of problems is a contribution by itself. Having the problem modeled in a reusable way is a base for implementing new ideas in future.

The modularity of the concept, the clear distinction between the model and the algorithm leaves room for separate independent corrections and enhancements. The concept of universality that results from the modularity is exceptionally useful. Having the model and the algorithm separated, they can both be replaced. The algorithm can be used for another problem, or the modeled problem can be solved with another algorithm, with only minor additional interventions. Enhancing and optimizing the clearly distinct model is easier. The separated model can be used for testing new algorithms and getting better results.

When building a solving engine, the universal algorithms and functions are implemented manage to give a solution. However, certain customizations and integration of small heuristic drastically accelerate the solving process. Adding guided search of the solution space showed significant improvement opposed to simple random solution proposal. However, a stochastic component is useful to avoid deadlocks and trapping in local optima.

By limiting the involvement of appropriate heuristic for the given problem type to modular components in the engine, the universality of the library is maintained. At the same time the performance is significantly increased. Finding a balance between universal optimizing functions and problem dependent heuristics, improves the engine for further more or less similar tasks.

7. References

Aarts, E. H. L., Korst, J. H. M., Laarhoven, P. J. M. V. (2003). *Simulated annealing In Local Search in Combinatorial Optimization*, Princeton University Press, ISBN: 0691115222, Princeton, New Jersey 08540 USA.

Abramson D. (1991) Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms, *Management Science*, Volume 37, Issue 1, pages 98 – 113

Bartak, R. (1999): Constraint Programming: In Pursuit of the Holy Grail, *Proceedings of WDS99*, Part IV, June 1999, pp. 555-564, MatFyzPress, Prague.

Blum C., Roli A., (2003) Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, Vol. 35, No. 3, pp. 268–308.

Blum C., Roli A., Alba E., (2005), *Parallel Metaheuristics*, *Wiley Book Series on Parallel and Distributed Computing*, John Wiley & Sons, ISBN: 9780471739388

Cave A., Nahavandi S., Kouzani A. (2002) Simulation Optimization for Process Scheduling through Simulated Annealing, *Proceedings of the 2002 Winter Simulation Conference*, 2002, San Diego, California, USA, pages 1269-1273.

Chorbev I., Dimitrovski I., Mihajlov D., Loskovska S. (2007) Hybrid Heuristics for Solving the Constraints Modeled High School Scheduling Problem, *Proc. of IEEE Region 8 Eurocon 2007 Conf.*, pages 2242-2249, ISBN: 978-1-4244-0813-9, Warsaw, Poland.

Chorbev I., Dimitrovski I., Loskovska S., Mihajlov D.(2006), A parallel implementation of Simulated annealing with a Boltzmann synchronization function and its application to solve the traveling salesman problem, Proc *IS2006*, pp 14-17, Ljubljana, Slovenija

Crawford B., Castro C., Monfroy E., (2007) Integration of Constraint Programming and Metaheuristics, *Lecture Notes in Computer Scienc, Abstraction, Reformulation, and Approximation,* Springer Berlin / Heidelberg, pp 397-398, ISBN 9783540735793

Czech Z., Wieczorek B.(2006) Solving bicriterion optimization problems by parallel simulated annealing, *Proceedings of the 14th Euromicro Workshop on Parallel, Distributed and Network –based Processing,* pp:7-14, ISBN ~ ISSN:1066-6192 , 0-7695-2513-X, 15-17 Feb. 2006, IEEE Computer Society Washington, DC, USA

Der-Rong Din, Tseng, S.S. (2001): Heuristic and Simulated Annealing Algorithms for Extended Cell Assignment Problem in Wireless ATM Network; *Journal of Information Science and Engineering*, Vol.17 No.4, pp.647-665

Duong T., Lam K. (2004), Combining Constraint Programming and Simulated Annealing on University Exam Timetabling, *Int. Conf. RIVF'04*, pages 205-210, Hanoi, Vietnam.

Gavanelli, M. (2002): Interactive Constraint Satisfaction Problems for Artificial Vision, in Ph.D. thesis, Department of Engineering, University of Ferrara, Italy.

Glover F. and M. Laguna (1997), *Tabu Search*, Kluwer Academic Publishers, Boston, ISBN 0 7923 8187 4.

Glover F., Laguna M. and Martí R., (2003), *Advances in Evolutionary Computation: Theory and Applications*, A. Ghosh and S. Tsutsui (Eds.), pp. 519-537, Springer-Verlag, ISBN:3-540-43330-9

Gomes N., Vale Z., Ramos C. (2005) Combining metaheuristics and constraint programming to solve a scheduling problem, *P. of the 4th WSEAS Int. Conf. on Applied Mathematics and Computer Science*, Article No. 5, ISBN:960-8457-17-3, Rio de Janeiro, Brazil, 2005

Hao J., Pannier J., (1998) Simulated Annealing and Tabu Search for Constraint Solving, in *Fifth Intl. Symposium on Artificial Intelligence and Mathematics*,

Hogg, T. and Huberman, A. (1993), Better than the best: The power of cooperation. *1992 Lectures in Complex Systems*, pp. 163–184, Addison-Wesley, Reading, MA.

Jolevski I., Loskovska S., Chorbev I., Mihajlov D., (a 2005) An Overview of a Constraint Solving Engine with Multiple Optimization Algorithms, *Proc. of the 27th International Conference on Information Technology Interfaces*, pp: 602- 608, ISBN: 953-7138-02-X, Cavtat, Croatia, June 20-23, 2005.

Jolevski I., Loskovska S., Chorbev I., Mihajlov D., Murgovski N., (c 2005) Constraints Modeling of the High School Scheduling Problem, *The Int. Conf. on Computer as a Tool, EUROCON 2005*, pp: 748-751, ISBN: 1-4244-0049-X, Belgrade, Serbia and Montenegro, 2005.

Jolevski I., Loskovska S., Chorbev I., Mihajlov D.,(b 2005) A Solution of N Queen Problem using a Constraint Solving Engine with Multiple Optimization Algorithms, *Proc. of ETAI 2005*, pages I56-I61, Ohrid, R. of Macedonia, 2005.

Jolevski I., Loskovska S., Murgovski N., Chorbev I., Mihajlov D.,(d 2005) Development of a user interface for a school scheduling application, ETAI, Ohrid, R. of Macedonia, *Proc. of ETAI 2005*, pages I4-4,I90–I95, Ohrid, R. of Macedonia, 2005.

Kumar, V. (1992): Algorithms for Constraint - Satisfaction Problems: A Survey, in *AI Magazine,* Volume 13, Issue 1, Spring 1992, pp: 32 - 44, American Association for Artificial Intelligence, ISSN:0738-4602, Menlo Park, CA, USA,

Leenen L., Venter L., Britz K.,(2003) A Pre-processing Algorithm for solving Constraint Satisfaction Optimization Problems, *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on*

Enablement through technology, p.11-15, ISBN:1-58113-774-5, September 17-19, 2003, South African Institute for Computer Scientists and Information Technologists

Meyer, M., (1994), Finite Domain Constraints: Declarativity meets Efficiency, in Doctor Dissertation

Ohlidal M., Schwarz J. (2004), Hybrid Parallel Simulated Annealing Using Genetic Operations, *Mendel 10th Internacional Conference on Soft Computing*, pp. 89-94, ISBN: 80-214-2676-4, Brno, Czech Republic, FSI VUT, 2004.

Penya Y. K., Jennings N. R., Neumann G. (2005) An Optimal Distributed Constraint Optimization Algorithm for Efficient Energy Management, *Proc. of Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, pp: 17- 182, ISBN: 0-7695-2504-0, 28-30 Nov. 2005, Addison-Wesley Publishing Company, Inc.

Tam V. and Ting D.,(2003) Combining the Min-Conflicts and Look-Forward Heuristics to Effectively Solve A Set of Hard University Timetabling Problems, *Proc. of the 15th IEEE International Conference on Tools with Artificial Intelligence*, p. 492, 1082-3409/03

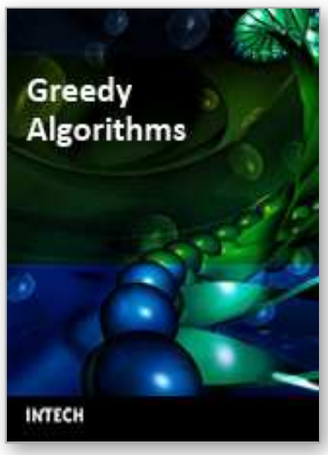
Tsang, Wang, Davenport, Voudouris & Lau.(1999) A family of stochastic methods for constraint satisfaction and optimization, *The First International Conference on The PACLP*, London, pages 359-383

Wah B., Chen Y.,(2001) Hybrid Constrained Simulated Annealing and Genetic Algorithms for Nonlinear Constrained Optimization, *Congress on Evolutionary Computation*, IEEE, pages. 925-932, Volume 2, 2001

Yoshikawa M., Kaneko K., Yamanouchi T., Watanabe M.,(1996) A Constraint Based High School Scheduling System, *Intelligent Systems and Their Applications*, pp 63 - 72, Vol. 11, Issue 1, ISSN:0885-9000, IEEE Educational Activities Department, NJ, USA

Zervoudakis K., Stamatopoulos P..(2001) A Generic Object-Oriented Constraint Based Model for University Course Timetabling, *Proc. of the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000,* Lecture Notes In Computer Science; Vol. 2079, pp 28 - 47, ISBN:3-540-42421-0, Springer-Verlag London, UK, 2000

Zhaohui F. and A. Lim, (2000) Heuristics for the Exam Scheduling Problem, *Proc. of the 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00),* pp 172-175, ISBN: 0-7695-0909-6.



Greedy Algorithms

Edited by Witold Bednorz

ISBN 978-953-7619-27-5
Hard cover, 586 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

Each chapter comprises a separate study on some optimization problem giving both an introductory look into the theory the problem comes from and some new developments invented by author(s). Usually some elementary knowledge is assumed, yet all the required facts are quoted mostly in examples, remarks or theorems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Ivan Chorbev, Suzana Loskovska, Ivica Dimitrovski and Dragan Mihajlov (2008). Solving the High School Scheduling Problem Modelled with Constraints Satisfaction Using Hybrid Heuristic Algorithms, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from: http://www.intechopen.com/books/greedy_algorithms/solving_the_high_school_scheduling_problem_modelled_with_constraints_satisfaction_using_hybrid_heuri

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen