Constraint-based Timetabling with Student Schedules

Hana Rudová and Luděk Matyska

Faculty of Informatics, Masaryk University
Botanická 68a, Brno 602 00, Czech Republic
{hanka,ludek}@fi.muni.cz

Abstract. One of the peculiarities of university timetabling problems lies in their huge complexity and the easy transition between complex constrained system and an over-constrained one. The Faculty of Informatics timetabling problem represents very complex scheduling and resource allocation problem as individual timetable for every student has to be scheduled with respect to heterogeneous choices of the students for their courses. The problem is solved via constraint programming approach, with variables' annotations for computing variable ordering and with special objective function aimed to minimize students' conflicts, both helping to solve over-constrained part of the problem as well as to control exploration of the search space.

1 Introduction

Faculty of Informatics timetabling problem represents large scale highly constrained scheduling problem as individual timetable for every student from more than thousand students has to be scheduled wrt. course pre-enrollment information. The number of prescribed courses is small, majority of courses are optional and so the sets of registered courses for each student can be very different. The probability of any two courses conflicting is about 30 % (which is equivalent to more then 10 000 constraints). Also some courses are taught in multiple sections and students must be assigned to sections while balancing section sizes and respecting room capacities (the so called student scheduling problem).

Timetabling problems may be solved by different methods inherited from operational research such as graph coloring and mathematical programming, from local search procedures such as tabu search and simulated annealing, or from genetic algorithms (see [25, 9] for surveys). Simulated annealing methods solve university-sized problems considering students preferences [12], assignment of students, times, and rooms to course sections was solved as series of assignment type problems [22]. Genetic [7] and evolutionary [10] algorithms were applied for student conflict minimization in examination planning.

Constraint logic programming (CLP) is particularly well suited for timetabling problems. It allows the formulation of all the constraints of the problem in a more declarative way than other approaches [18, 16]. CLP was applied for various course timetabling problems [1, 2, 13, 15, 16, 17, 18] but none of them includes search for schedules of individual students. To our knowledge, any constraint programming system creating automated timetable for individual students or defining similar objective function has not yet been described. Similarities may be found with examination timetabling [5] but problem definition is quite different and it require (almost) conflict free schedules which is not feasible for our problem at all. In this paper, we would like to show how CLP's approach may be applied even for such type of problems.

The first part of the paper will overview techniques applied for solving constraint satisfaction problems with preferences, show their real application in timetabling and describe variables' annotations — preferences of particular variables in constraints. Next sections describes our timetabling problem together with its decomposition into student scheduling, classroom allocation and time assignment subproblems. As a part of time assignment, we will propose objective function aimed to minimize conflicts between particular courses for individual students. This objective function allows us to compute cost for each particular course time assignment and evaluate achieved solution quality. Computational results together with possible further improvements conclude the paper.

2 Constraints with Preferences in Timetabling

Various types of preferences, priorities, satisfaction degrees, or weights were proposed to find solutions of problems with uncertainties, ill-defined problems, optimization problems, and over-constrained problems where some kind of softness have to be involved to get feasible solutions. Preferences are either assigned to constraints [14, 6, 26] or to each tuple of values for a constraint [11, 4]. The most simple framework, the maximal constraint satisfaction (MAX-CSP) [14] seeks a solution that satisfies as many constraints as possible. Weighted CSP considers weights for each constraint and minimizes weighted sum of unsatisfied constraints. Fuzzy CSP [11] combines preference degrees with help of fuzzy sets and possibility theory. Constraint hierarchies [6] define several levels of constraints, the violations of constraints are minimized level by level subsequently with help of weighting the importance of constraints. Valued CSP [26] and Semiring-based CSP [4] represent meta-frameworks defining monoid and lattice structures, which specify type of used preferences and manipulation with them.

The complexity of these systems is still high and a lot of work remains to be done for their application in real-life large scale problems. The crucial part of the problem solution becomes also global constraints which were efficiently implemented as a soft constraints only for minor sub-problems [3, 21]. In the following part, we will give an overview of timetabling papers which tries to include preferences into the problem solution. Applied methods are less complex than those presented above but they help significantly improve solution quality.

Within the traditional CSP approach, preferences for selecting feasible solution are implemented with the help of some labelling heuristics [2, 15]. Golz et al. [15] applies the typical solutions — given unary soft constraints with priority are integrated into the solution search through value and variable ordering heuristics. Azavedo et al. [2] describe decision steps which lead to computing final labelling heuristics, which instantiate hard variables first, e.g., courses with demanded classrooms, or longer durations.

An expensive search of overall solution space may be replaced by relaxation of difficult constraints either in a semi-automatic [18] or an automatic way [16].

An optimization constraint was applied for solving of medium-sized problems [13, 17]. While the first paper [13] describes construction of optimum timetable according to a given cost function, Henz et al. [17] recognize that going for the globally best solution is not feasible. When the size (120/90 courses for [13]/[17], resp.) and type (classical course timetabling without student schedules) of the solved problems are considered, it remains doubtful whether this approach would scale up.

Abdennadher et al. [1] describe interesting implementation having soft constraints as an active part of constraint propagation with help of cost-based approach collecting a cost for each value in the domain of time variable. However, the solution was applied for the department-sized problem only.

Our aim is to propose variable ordering heuristics which would be able to catch user's preferences together with his expectations about difficulty of variable's instantiation which seems to be crucial from the previous discussion about papers [2, 15]. We would like also to express preferences on non-unary constraints and reflect them into the problem solution. These ideas tries to realize variables' annotations which will be described in the next section. The second method which will include user's preferences on timetabling problem is the definition of suitable objective function whose value could be sub-optimized at least. In Sec. 4.3, we propose an objective function which controls satisfaction of particular student schedules. Combining of both these approaches help us to solve over-constrained part of the problem as well as to control exploration of the search space.

2.1 Constraints with Variables' Annotations

We will apply variables' annotations [24, 23] to overcome difficulties given by solving over-constrained problems or problems with large search spaces. Let us emphasize that we consider neither preferences of constraints [26] nor costs on tuples of values for constraints [4], annotation just represent preference of each variable in particular constraint. Annotations are given by user as a part of input data and they are applied for computing variable ordering.

Annotations seems to be suitable for timetabling problems with already preordered variables representing courses (more and less engaged by students), classrooms (different size, different equipment, ...), and teachers (dean, professors, assistants, ...). The more important annotations may reflect user's preferences of some variables above others, e.g., time for courses with larger number of students or taught by some senior professor. Annotations may also express user's opinion on difficulty of variable's instantiation following from the complexity or character of the constraint, e.g., requirement for scarce resources, strong time dependencies between several course sections, or strong requirements for teaching time of some course section. Such annotated variables may be a source of propagation and their early instantiation may prune the solution space or avoid mistakes both at the beginning of the search.

Let us imagine a simple constraint which requires to teach one course (`T1`) before the another (`T2`). The first course should be taught in a more occupied classroom than the second. This relation may express the following constraint with annotations: `T1@strong < T2@weak`. The annotation `strong` become a source of early instantiation of the variable `T1`. If it occurs, the constraint is satisfied and a reduction of domain of `T2` is less critical than would be a reduction for the variable `T1` wrt. stronger requirement on `T1`. These annotations may also express that `T1` is taught by some professor and `T2` by an assistant. Such annotations become very interesting when considered together with preferential constraints for value ordering which restrain early morning and late evening hours. Any later instantiation would probably worse cost of value selected from domain of variable.

Let us consider constraint satisfaction problem (CSP) as a tuple (V, D, C) with set of variables V, domains of variables in D and set of constraints C. Annotations are defined by a triple $(\mathcal{A}, \preceq, \circledast)$

- $\mathcal{A}$ as a set of annotations;
- $\preceq$ as an ordering on $\mathcal{A}$;
  if $a, b \in \mathcal{A}$ then $a \preceq b$ means a is more preferred annotation than b;
- $\circledast_{a_i \in A} a_i$ (finite $A \subset \mathcal{A}$) as a function computing global annotation; it is defined by applying either $\circledast : \mathcal{A}^k \rightarrow \mathcal{A}$ or a commutative and associative binary operation closed on $\mathcal{A}$.

Annotations are local to a constraint, i.e., any variable may have different annotations in different constraints. Function $a : C \times V \rightarrow \mathcal{A}$ determines the annotation of every variable in every constraint. Local variable's annotation is used for computing several characteristics of underlying constraint system, the so called global annotations. Let us define $var(c) \subseteq V$ as a set of variables in the constraint c. Then we can define global variable annotation by combining variable's annotations in all constraints where variable occurs

$$av : V \rightarrow \mathcal{A} \ , \ \ av(v) = \underset{\{c \in C \mid v \in var(c)\}}{\circledast} a(c, v) \ .$$

During computation particular variables are instantiated and constraints become non-active due to their (un)satisfaction. Let us denote V_{act} as a set of still uninstantiated variables, C_{act} as a set of still unresolved[1] constraints in some computation step. The set of uninstantiated variables in the constraint c is denoted $var_{act}(c) \subseteq V_{act}$. We define dynamic global variable annotation, which allows us to consider the actual state of computation

$$dav : V_{act} \rightarrow \mathcal{A} \ , \ \ dav(v) = \underset{\{c \in C_{act} \mid v \in var_{act}(c)\}}{\circledast} a(c, v) \ .$$

[1] Unresolved constraint is still neither true nor false. False constraints may be soft constraints only.

Currently, global variable annotations are applied for computing dynamic or static variable ordering which is combined with value ordering given by value preferences (i.e., annotations) of particular variables.

Typical examples of $(\mathcal{A}, \preceq, \circledast)$ are

$\mathcal{A}_{sum} = (N, \geq_N, +)$;
$\mathcal{A}_{avg} = (\langle 0, 1\rangle \subseteq R, \geq_R,$ arithmetic average);
$\mathcal{A}_{hier} = (N^k, \geq_{lex}, +_{vect})$

- fixed $k \in N$;
- $\geq_{lex}$ is standard lexicographic ordering;
- $+_{vect}$ is standard vector sum;
- $a(c, v) = (0, \dots, 0, w, 0, \dots, 0)$.

$\mathcal{A}_{sum}$ allows weighted approach to particular annotations, the variable with the highest sum of annotations is selected as the first for instantiation. $\mathcal{A}_{avg}$ selects the variable with the highest average annotation. While $\mathcal{A}_{sum}$ allows only one level of weighted annotations, $\mathcal{A}_{hier}$ considers k weighted levels of hierarchy. These levels are aimed to express strict requirements on instantiation order of some variables, e.g., colloquy visited by most teachers should have instantiated starting time first while instantiation of starting time for the course with one teacher may be postponed.

3 Problem Description

Our university timetabling problem is a bit more complex than usual, because we would like to create individual timetable for every student — as (s)he registers for a set of courses. Also the number of prescribed courses is small, majority of courses are optional and so the sets of registered courses for each student can be very different. Still higher level of complexity adds another problem: each course may consists from lecture or seminar or lecture+seminar, where the number of lectures and/or seminars is determined by the number of students pre-enrolled on a course, and with respect to teacher's requirements. Students must be assigned to course sections (particular seminar or lecture) while balancing section sizes and respecting room capacities. Computing an ideal timetable should solve the problem of student scheduling such that each student is able to visit one lecture and/or one seminar for each his/her specified course. Each course section is then given by a tuple ⟨course, lecture or seminar identifier/order, set of students⟩. Basically our task consists in instantiation of the set of tuples ⟨course section, classroom, time⟩, i.e., each lecture or seminar of course has assigned its set of students, classroom, and time. Teachers of particular courses are determined as a part of our problem definition, so we do not need to solve this kind of resource allocation — assign teachers to courses.

The solution of overall problem should consider the following subproblems (not necessarily subsequently in prescribed order).

Student scheduling assigns students to course sections, where an input of this problem are tuples ⟨number of lectures, number of seminars, students⟩ given for each course, and an output is a set of course sections defined by tuple ⟨course, lecture or seminar identifier, set of students⟩.

Classroom allocation for particular course sections while satisfying following hard constraints

- (C1) only one course can take place in a classroom at a time;
- (C2) each classroom has to have sufficient size wrt. expected number of students in course section;
- (C3) each classroom has to have suitable equipment due to teacher's requirements (lecture hall, computer room, lecture hall with data projector);
- (C4) exception — one teacher can teach one course in two specific classrooms at the same time either with help of camera and video projection or to ensure arbitrary exchange between computer room and lecture hall;

Time assignment of starting times to course sections while minimizing a total number of course sections which overlap for any student and satisfying following hard and soft constraints: (C1), (C4), and

- (C5) each teacher gives only one course section at a time;
- (C6) maximal number of time slices for teacher per day has to be respected;
- (C7) requested time of courses wrt. strict unavailability of teacher;
- (C8) various hard time dependencies between two or more course sections;
- (C9) Friday afternoon is restrained;
- (C10) early morning and late evening times are generally restrained;
- (C11) special time preferences for course section specified by teacher;
- (C12) various soft time dependencies between two or more course sections given by teachers.

Problem Size. With respect to growing size of our school we can use input data of different scope. Each instance is timetabled for 5 days with 13 teaching time slices. Each course section is non-breakable and it has variable duration from 1 to 4 time slices. Currently our implementation includes two different data sets with courses scheduled at the main faculty building. The number of our course sections is 234 and 269, resp. We have 93 (113) teachers, 12 (18) classrooms, 948 (1 255) pre-enrolled students[2], and 10 360 (13 289) student's requirements in course pre-enrollment, resp. Currently our problem definition includes 454 and 483 preferential requirements, resp. The probability of any two course sections conflicting is about 30 % (which is equivalent to 7 440 and 11 635 constraints, resp.).

4 Problem Solution

Generally all subproblems should be solved together to obtain such assignment of students, classrooms, and times to course sections that the hard constraints are

[2] Real number of students is slightly higher — about 1 000 and 1 400 students.

satisfied, the objective function is minimized, and all soft constraints are satisfied to the largest possible extent. The complexity of particular subproblems is very high — especially student scheduling problem increases it so far that it remains doubtful whether solution of such a large scale highly constrained problem would be found in any reasonable time.

We separated the most complex student scheduling problem and we constructed the feasible division of students as an input for other problems. This approach allows us to compute for every two course sections cs_1, cs_2 a number $\mathit{TwoConflict}(cs_1, cs_2)$ of students which want to visit both course sections. With this we know how desirable or undesirable is parallel teaching of any two course sections. The objective function summarizes $\mathit{TwoConflict}(cs_1, cs_2)$ for any overlapping course sections cs_1, cs_2. This value of objective function computed for one pre-solved division of students gives us the worst case cost estimation of generated timetable — some places in lectures/seminars are still free with respect to existing overlap for some students and students are also allowed to swap lectures/seminars with other students to improve their personal schedules.

With pre-solved student scheduling problem, our effort will be devoted to an instantiation of two types of variables — time assignment variable and classroom variable for each course section.

The above proposed solution may include hard requirements as standard constraints in constraint satisfaction problem, but enlarged set of constraints including all the preference requirements (C9)+(C10)+(C11)+(C12) would make the problem over-constrained. This part of problem may be understood as the second objective which should evaluate "better" placement of particular course sections measured by "better" satisfaction of preferential (soft) constraints. Controlled exploration of the search space during time assignment is allowed by variables' annotations which also help to satisfy suitable soft constraints.

We started our implementation in SICStus Prolog CLP(*FD*) Solver [8] based on global constraints as `cumulative`, `serialized`, and `all_distinct` for sharing cumulative and unary resources. This implementation did not include all hard constraints and neither soft constraints nor objective function were a part of problem representation, but it was not very efficient — the first solution was found approximately in 1 minute on a Pentium II/400 MHz PC. We tried to compare this version with implementation in ILOG software [19, 20], an object oriented library for constraint logic programming in C++. Because the first obtained solution was much faster (1 second on the same system), for the second version which will be described in the following part we used the ILOG environment only.

4.1 Student Scheduling

The output of this problem should be some feasible splitting of students into particular course sections. Set of registered students for each course is basically sorted in lexicographic order and then split wrt. the required number of groups. The lexicographic order is advantageous wrt. the splitting of the same student group to lectures and to seminars, i.e., useless conflicts for seminars and lectures of the same course are not imposed.

4.2 Classroom Allocation

Each classroom of unique size plays a role of unary resource (`IlcUnaryResource` in ILOG), rooms of the same size are grouped together into one cumulative resource (`IlcDiscreteResource`) with capacity equal to the number n of classrooms. This approach postpones some parts of resource allocation after time assignment which may be crucial for finding solution in acceptable time. This also allows us to decrease a number of symmetries in CSP. Particular classrooms are assigned after solving the CSP. Existence of a solution is guaranteed even for non-breakable course sections and this solution is found in $O(n \times time)$. The construction of such solution shows the following proof.

Proof. ILOG object `IlcDiscreteResource` representing sets of classrooms of the same size ensures that smaller or equal number of timetabling items than its capacity (the number of classrooms in the set) is allocated in every time. Now let us take classrooms in any fixed order. Classrooms are assigned to course sections subsequently. Each classroom is assigned to course section in increasing time without any gaps if possible. When some course section have remained without assigned classrooms in the end, then some gap beginning at the starting time of course section has to exist wrt. existing cumulative resource. But all course section were assigned in order given by starting time and any such gap should not exist — the contradiction compelling the proof. □

Each course sections has to have assigned classroom of suitable size and suitable equipment. Currently we have three different types of classrooms — lecture halls, lecture halls with data projector, and computer rooms — each of them with several different sizes. We have created three "hierarchies" of classrooms, the highest level always contains the largest classrooms while the lowest level contains all classrooms. Each course section `requires` classroom from the sufficient level (C2) and requested room hierarchy — type (C3). Particular levels represent sets of alternative resources (`IlcAltResSet`). Only course sections which require specific classroom are directly assigned to them (C4).

Possible classrooms are partially assigned to course sections prior to time assignments. Each classroom hierarchy is independent on others, because hierarchies are either too different or over-full by their own course sections. Classroom variables are assigned for each hierarchy in order given by required room size. Less engaged course sections are assigned last to ensure available classrooms of sufficient size for more engaged course sections. Exact distribution of classrooms of the same size (`IlcDiscreteResource`) is the last completed. Last section will discuss possible improvements of this solution.

4.3 Time Assignment

Hard Constraints. Each teacher is represented by unary resources (ILOG object `IlcUnaryResource`). Course sections are basically represented by their starting time and duration because they are not allowed to be interrupted (`IlcIntervalActivity`). Each course section `requires` suitable teacher (C5). This constraint adds annotation for time of course section corresponding to the seniority of the teacher. Constraint (C6) restrains the number of time slices for teacher per day by `IlcDiscreteEnergy`. The value of time slices is related with annotation — more important annotation corresponds to the lower number of time slices. Some teachers may prohibit or order particular teaching times (C7) and annotation express how strong such requirement is. Other required constraints (C8) are implemented with help of ILOG predefined constraints.

Course sections taught by one teacher at the same time (C4) may be easily solved by removing the constraint (C5) for one of the course sections and binding them to the same time. Annotations of both time variables in constraint (C4) depends on type and size of demanded classroom for each course section, resp.

Soft Constraints. Preferential constraints are integrated into the solution search. Some of them are posted (if possible) to prune the solution space as a consequence of assignment of variable which occurs in the constraint. Some (usually unary) constraints takes a role of checkers — they are satisfied if satisfactory assignment exists. However annotation indicates each time which variables have to be taken care of.

Unary constraints usually prefer times from 9 a.m. to 5 p.m. (C10) or restrain Friday's afternoon (C9), but particular value ordering may be specific wrt. teacher's requirement (C11). Annotation of variables in all these constraints stress more frequent times and smaller required domains.

Time dependency constraint (C8) includes non-standard requirements of particular teachers. Set of course sections should be taught in the same day, some lectures should be taught before seminar. Annotations in these constraints represent seniority of teacher. Special colloquy taught by invited lecturers would like to attend most teachers. Annotation of starting time of colloquy in this constraint belongs to the most important annotations. Later assignment of this variable would be very problematic. This constraint also includes times of all course sections taught by those teachers. Their annotation is again given by the seniority of each teacher. Teaching (PhD.) students may prefer non-overlapping of several courses. Some non-overlapping courses are specified as required constraints but most of them include student priority to satisfy such requirement. This priority is directly translated into annotation of particular course section.

Objective Function. Optimization criteria tries to minimize sum of course sections, which student is not able to attend wrt. concurrent schedule of other his (her) course sections, for all students. Each generated timetable may be evaluated by so called *StudentsConflict* reflecting solution quality wrt. the value of objective function. The following part defines *StudentsConflict* and its approximation and describes the process of computing sub-optimal solution (time assignment subproblem) with help of *StudentsConflict* approximation.

Each student pre-enrolls some set of courses. Student scheduling assigns fixed course section(s) for each his (her) course. Because we don't know which of conflicting course sections will be selected by particular students, we need to consider some selection criteria c to express course sections attendance by each student. $\mathit{StudentsConflict}(c)$ will summarize the number of course sections which students are not able to attend due to selection criteria c.

$$\mathit{StudentsConflict}(c) = \sum_{s} \mathit{ConflictingSection}(c, s) \tag{1}$$

$\mathit{ConflictingSection}(c, s)$ denotes the number of course sections which student s is not able to attend wrt. selection criteria c. To obtain correct interpretation of solution quality, we have to consider $\mathit{StudentsConflict}(c_{max})$ with worst possible selection of course sections by students as our cost estimation.

$$\exists c_{max} \forall c \; \mathit{StudentsConflict}(c_{max}) \geq \mathit{StudentsConflict}(c)$$

Example 1. Let us demonstrate above statements by a small example. We will consider contribution of one student to the sum *StudentsConflict*. Selected student is assigned to course sections {p084, v038, p086}. Small example of generated timetable is shown at Fig. 1. Student s may decide to cancel either course

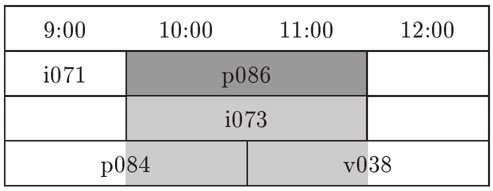

Fig. 1. Small example of generated timetable

section p086 ($\mathit{ConflictingSection}(c_1, s) = 1$) or course sections p084 and v038 ($\mathit{ConflictingSection}(c_2, s) = 2$). $\mathit{StudentsConflict}(c_{max})$ has to subsume the worst case contribution for each student, which means $c_{max} = c_2$.

Let us define $\mathit{TwoConflict}(i, j)$ as a number of students which pre-enrolls both course sections i and j. For course section i, $\mathit{TimeOverlap}(i)$ is a set of all course sections having overlap with i, $\mathit{VarOrder}(i)$ gives instantiation order of time assignment variable for course section i in the sequence of time assignment variables.

Let us denote $\mathit{StudentsConflict}_{inc}$ as our approximation of cost estimation $\mathit{StudentsConflict}(c_{max})$ which is incrementally computed and optimized during time assignment of particular course sections i.

$$\mathit{StudentsConflict}_{inc} = \sum_{i} \mathit{SectionConflict}(i)$$

Each *SectionConflict*(i) contributes to the resulting *StudentsConflict*$_{inc}$ by sum of *TwoConflict*(i, j) for each course section j already scheduled (*VarOrder*$(j) <$ *VarOrder*(i)) and overlapping with i ($j \in$ *TimeOverlap*(i)).

$$SectionConflict(i) = \sum_{\substack{j \in TimeOverlap(i) \\ VarOrder(j) < VarOrder(i)}} TwoConflict(i, j)$$

Example 2. Let us go back to the timetable in Fig. 1 and suppose that times of course sections were assigned in order p084, v038, i073, i071, and p086. For example *SectionConflict*(v038) is equal to 0 because none of previously assigned course sections (only p084) overlaps with v038. Last assigned course section p086 overlaps with p084, v038, and i073. Its *SectionConflict* is equal to the sum *TwoConflict*(p086,p084) + *TwoConflict*(p086,v038) + *TwoConflict*(p086,i073).

Corollary 1. *The value StudentsConflict$_{inc}$ is an upper bound of StudentsConflict(c) for any selection criteria c, i.e.,*

$$StudentsConflict_{inc} \geq StudentsConflict(c_{max}) \ .$$

Proof. *StudentsConflict*(c_{max}) summarizes *ConflictingSection*(c_{max}, s) for all students s (see Eqn. 1). *ConflictingSection*(c_{max}, s) includes some selection of overlapping course sections cs_i for $i = 1 \ldots$ *ConflictingSection*(c_{max}, s) of student s. Each cs_i is subsumed into *ConflictingSection*(c_{max}, s) if it overlaps with some course section cs_{io} also pre-enrolled by student s. This overlap is included either in *SectionConflict*(cs_i) for *VarOrder*$(cs_{io}) <$ *VarOrder*(cs_i) or in *SectionConflict*(cs_{io}) for *VarOrder*$(cs_i) <$ *VarOrder*(cs_{io}). □

StudentsConflict$_{inc}$ also cumulates some additional fail contributions. These contributions are related to multiple course sections of one student overlapping at the same time.

Example 3. Let us consider a student attending courses sections {p084, v038, i073, p086} and the timetable from the Fig. 1. A contribution to the objective function *StudentsConflict*(c_{max}) corresponds to 3 but its approximation *StudentsConflict*$_{inc}$ is increased by 5 for any ordering of course sections during time assignment.

The resulting quality of our upper bound depends both on selection criteria c and on above described cumulative error. However, these inaccuracies decrease with increasing quality of generated timetable and may be even abandoned for sufficiently fair solutions. To show possible differences between upper bound *StudentsConflict*$_{inc}$ and *StudentsConflict*(c), we will also present results for selection criteria *VarOrder*—each student attends those of conflicting course sections which have earlier instantiation order of their time assignment variable.

Generated timetables for different input data are compared with help of *Ratio*$_{inc}$ and *Ratio*(*VarOrder*)

$$Ratio_{inc} = \frac{StudentsConflict_{inc}}{NbStudentRequirements} \ , \quad Ratio(VarOrder) = \frac{StudentsConflict(VarOrder)}{NbStudentRequirements}$$

where *NbStudentRequirements* is a number of student's requirements in course pre-enrollment.

Variable and Value Ordering. Time assignment variables are instantiated in order given by their global variable annotation (see Sec. 2.1). The two possible instances of annotation framework are considered in our problem. Basically $\mathcal{A}_{hier}$ aapproach puts variables in teacher's requirements on the higher level while student's interests or classroom properties are deferred. $\mathcal{A}_{sum}$ summarizes all annotations in one level. However, both solutions slightly support some variables in constraints, which may be difficult to satisfy (e.g., starting time of special colloquy attended by most teachers).

Each time a variable is instantiated special value ordering is imposed. This value ordering depends on the preferred values in the domain of particular time variable for course section i and on the number *SectionConflict*(i). Several levels of preferences may be given by soft constraints. The preferred values from the first level are selected in increasing order and current value of overlaps *SectionConflict*(i) is compared with two kinds of thresholds. Absolute threshold is a maximal number of students in one course section for whom some overlapping with other course sections may occur and relative threshold is a ratio between the number of students with any overlap and the number of students pre-enrolled to the course section *NumberStudents*(i)

- Absolute threshold: $\mathit{SectionConflict}(i) \leq \mathit{AbsThreshold}$
- Relative threshold: $\frac{\mathit{SectionConflict}(i)}{\mathit{NumberStudents}(i)} \leq \mathit{RelThreshold}$

If *SectionConflict*(i) is not sufficient the next remaining value is selected. Both thresholds may be also combined — if one of the thresholds is satisfied, value may be selected for instantiation. If values of current preference level are not sufficient to find feasible solution, values from lower level are tried in increasing order again. Possible faulty instantiations are undone by backtracking.

5 Computational Results

Computing the first solution takes 2–3, and 4–5 seconds on a Pentium II/400 MHz PC for the first and the second problem instance (particular data sets were described in Sec. 3), resp. Table 1 shows achieved results for particular data sets and annotation frameworks (general description of annotations may be found in Sec. 2.1; for further information about the concrete instances $\mathcal{A}_{sum}$ and $\mathcal{A}_{hier}$ see Sec. 4.3). The best achieved values of thresholds *AbsThreshold* and *RelThreshold* (both defined in Sec. 4.3) follow in the third and fourth columns. For the presented thresholds, generated solutions are evaluated also with help of the percentage of unsatisfied soft constraints and with help of Ratio_{inc} together with *Ratio*(*VarOrder*) (for definitions see Sec. 4.3) giving quality of solution wrt. the objective function.

$\mathcal{A}_{sum}$ framework computes better solution for absolute thresholds while $\mathcal{A}_{hier}$ annotations output more interesting results with relative thresholds. This is

Table 1. Computational results for timetabling problem at FI

Data set	$\mathcal{A}$	*AbsThreshold*	*RelThreshold* (%)	Unsatisfied soft constraints (%)	$Ratio_{inc}$ (%)	*Ratio* (*VarOrder*) (%)
I.	$\mathcal{A}_{sum}$	4	–	7.3	4.6	4.5
I.	$\mathcal{A}_{hier}$	–	9	8.1	5.2	5.1
I.	$\mathcal{A}_{sum}$	3	6	9.7	3.6	3.5
I.	$\mathcal{A}_{hier}$	1	7	8.8	4.4	4.3
II.	$\mathcal{A}_{sum}$	8	–	5.0	7.9	7.7
II.	$\mathcal{A}_{hier}$	–	14	8.1	6.3	6.1
II.	$\mathcal{A}_{sum}$	3	13	9.3	5.5	5.3
II.	$\mathcal{A}_{hier}$	2	13	8.1	6.0	5.8

a consequence of higher support of student's interests expressed by the number of students in $\mathcal{A}_{sum}$. While the large course sections (up to 258 students) were satisfied to the high extent, importance of smaller course sections (at least 16 students) was deferred. If only one of the both thresholds have to be satisfied, results for $\mathcal{A}_{sum}$ and $\mathcal{A}_{hier}$ were comparable. Results also show that both optimization criteria are antagonistic — if student's conflicts are minimized, the quality of satisfaction of preferential constraints decreases, and vice versa.

6 Conclusion

We have successfully implemented timetabling problem at the Faculty of Informatics reflecting large scale course pre-enrollment requirements. We have shown that constraint programming approach may be applied even for such a highly constrained problem. Constraint programming together with annotations allows us easy incremental addition of other requirements without changing general solving strategy. This statement was verified not only by extending the basic set of constraints but also by later inclusion of the more extensive data set.

Currently achieved results with more than 94 % of satisfied requirements in course pre-enrollment in the first computed solution promise interesting results even with additional constraints. These results seems to be very interesting when compared with faculty initial requirement to satisfy 60–70 % course pre-enrollments.

Our further improvements will include classroom allocation. We would like to balance room occupation not only for rooms of the same size but also for all rooms of the same type to increase degree of freedom in the time assignment. Currently annotations influence labelling process of time assignment variables, we would like to extend their importance for classroom assignment variables. Some classroom requirements should be also added as soft or hard constraints with annotations which should take a role in the solution search. Other possibility consists in interleaved classroom and time assignment directed by annotations. We also plan further improvements of the search method during instantiation of time assignment variables.

Annotations seems to be well suited for solving timetabling problems where the preferences on variables are easily expressed. Our future work will study potential of the annotations for solving constraint satisfaction, optimization, and over-constrained problems wrt. their proposed possible semantics. We would like to extend their importance to a general solving strategy for problems with partially or even completely ordered variables.

Acknowledgements

We would like to thank Libor Škarvada for the discussions about current creation of the timetable at the Faculty of Informatics.

References

[1] Slim Abdennadher and Michael Marte. University timetabling using constraint handling rules. In *Actes des Journées Francophones de Programmation en Logique et Programmation par Contraintes*, 1998.

[2] Francisco Azevedo and Pedro Manuel Barahona. Timetabling in constraint logic programming. In *Proceedings of the 2nd World Congress on Expert Systems*, 1994.

[3] Philippe Baptiste, Claude Le Pape, and Laurent Péridy. Global constraints for partials CSPs: A case-study of resource and due date constraints. In Michael Maher and Jean-François Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 87–101. Springer-Verlag LNCS 1520, 1998.

[4] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.

[5] Patrice Boizumault, Yan Delon, and Laurent Péridy. Solving a real-life planning exams problem using constraint logic programming. In Manfred Meyer, editor, *Constraint Processing: Proceedings of the International Workshop at CSAM'93*, Research Report RR-93-39, pages 107–112, DFKI Kaiserslautern, August 1993.

[6] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[7] Edmund K. Burke, Dave G. Elliman, and Rupert F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 605–610. Morgan Kaufmann, July 1995.

[8] Mats Carlsson, Greger Ottosson, and Björn Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programming*. Springer-Verlag LNCS 1292, 1997.

[9] Michael W. Carter and Gilbert Laporte. Resent developments in practical course timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 3–19. Springer-Verlag LNCS 1408, 1998.

[10] David Corne and Peter Ross. Peckish initialisation strategies for evolutionary timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 227–240. Springer-Verlag LNCS 1153, 1996.

[11] Didier Dubois, Hélène Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[12] M. A. Saleh Elmohamed, Paul Coddington, and Geoffrey Fox. A comparison of annealing techniques for academic course scheduling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 92–112. Springer-Verlag LNCS 1408, 1998.

[13] Harikleia Frangouli, Vassilis Harmandas, and Panagiotis Stamatopoulos. UTSE: Construction of optimum timetables for university courses — A CLP based approach. In *Proceedings of the Third International Conference on the Practical Applications of Prolog (PAP'95)*, pages 225–243, Paris, France, April 1995. Alinmead Software Ltd.

[14] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[15] Hans-Joachim Goltz, Georg Küchler, and Dirk Matzke. Constraint-based timetabling for universities. In *Proceedings INAP'98, 11th International Conference on Applications of Prolog*, pages 75–80, 1998.

[16] Christelle Guéret, Narendra Jussien, Patrice Boizumault, and Christian Prins. Building university timetables using constraint logic programming. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 130–145. Springer-Verlag LNCS 1153, 1996.

[17] Martin Henz and Jörg Würtz. Using Oz for college timetabling. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 162–177. Springer-Verlag LNCS 1153, 1996.

[18] Gyuri Lajos. Complete university modular timetabling using constraint logic programming. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 146–161. Springer-Verlag LNCS 1153, 1996.

[19] Claude Le Pape. Implementation of resource constraints in ILOG SCHEDULE: a library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[20] Jean-François Puget. A C++ implementation of CLP. Technical report, ILOG S.A., 1994.

[21] Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming — CP'99*, pages 390–404. Springer-Verlag LNCS 1713, 1999.

[22] Vincent Robert and Alain Hertz. How to decompose constrained course scheduling problems into easier assignment type subproblems. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling*, pages 364–373. Springer-Verlag LNCS 1153, 1996.

[23] Hana Rudová. Constraints with variables' annotations. In Henri Prade, editor, *13th European Conference on Artificial Intelligence Proceedings*, pages 261–262. John Wiley & Sons, Ltd., 1998.

[24] Hana Rudová and Luděk Matyska. Uniform framework for solving over-constrained and optimization problems. In *CP'99 Post-Conference Workshop on Modelling and Solving Soft Constraints*, 1999.

[25] Andrea Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI, Amsterdam, NL, 1995.

[26] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639, San Mateo, August 1995. Morgan Kaufmann.