FI MU

Faculty of Informatics
Masaryk University

Timetabling with Annotations

by

Hana Rudová
Luděk Matyska

FI MU Report Series FIMU-RS-99-09

 December 1999

Timetabling with Annotations

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

December 21, 1999

Abstract

One of the peculiarities of university timetabling problems lies in their huge complexity and the easy transition between complex constrained system to 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 course pre-enrollment informations. Variables' annotations were proposed to define preferences of variables in constraints and they serve as a source for computing variable ordering in optimization problems where the search space is too large to be fully traversed and explored. Annotations suggest a route through this space which leads quickly to at least sub-optimal solutions. Annotations may even help to find preferred solutions first as they instantiate preferred values in the domain of variables as soon as possible.

1 Introduction

Variables' annotations express preferences of variables in particular constraints. They are applied for solving constraint satisfaction problems with too large search space to be fully traversed and explored. Annotations could be suitable for application areas such as planning or scheduling. Let us consider the timetabling problem stated in variables which represent teachers (dean, professors, assistants, ...), rooms (different size, different equipment, ...), and courses (more and less engaged by students). These various degrees of importance define annotations which may be stated directly as a part of input data. If a problem has too large number of solutions to explore them all then annotations are used to generate more interesting solutions as soon as possible.

Another interesting example is the problem of the search for the optimal sequence of aircraft departures from a runway. Because practically all constraints are safety regulations, we can only change aircraft allocation to different time slots. The only allowed action is the removal of an aircraft from further consideration. We can assign preferences to individual variables (planes) and find solutions with more important aircrafts in preferred time slots prior to other solutions.

Our paper consists from two main parts, the first one describes annotations as an approach for solving constraint satisfaction problems with optimizations and the second part is devoted to the real university timetabling problem solved with help of annotations. Both parts include summary of related work concerning systems with different kind of preferences applied for solving constraint satisfaction problems and methods for solving timetabling problem as the constraint satisfaction problem.

2 Annotations

Our former work with annotations introduces them as an approach for solving over-constrained problems. These annotations represented importance or meaning of variable's occurence in constraint to define selection of feasible solution in problems, where all constraints can not be satisfied. We have defined mappings of variables' annotations to different frameworks [Rud98b, Rud98c, Rud98a] for solving over-constrained problems — to constraint hierarchies [BFBW92, WB93] and possibilistic CSPs [Sch92]. Particular mappings give us several interpretations of annotations and they may be used as examples of possible semantics of annotations in over-constrained problems.

This work applies the same preferences as above for solving constraint satisfaction problems with optimizations where the whole search space can not be explored due to its complexity. Here annotations define a route through the search space leading to at least sub-optimal solution. Preferences of variables in particular constraints will define variable ordering, which is combined with value ordering heuristics.

2.1 Related Work on Preferences

Various types of preferences, priorities, satisfaction degrees, weights, or levels of importance were proposed to find solutions of problems with uncertainties, ill-defined problems, over-constrained problems, and optimization problems where some kind of softness have to be involved to get feasible solutions. The most simple framework, the maximal constraint satisfaction (MAX-CSP) [FW92] seeks a solution that satisfies as many constraints as possible. Weighted constraint satisfaction considers weights for each constraint and minimizes weighted sum of unsatisfied constraints. Both of these systems are used for solving over-constrained problems. Possibilistic CSP [Sch92] assigns to each constraint some preference degree, which express necessity of its satisfaction. Fuzzy CSP [DFP96] considers constraint as a relation assigning to each tuple of values its level of preferences. Preference degrees in both approaches are combined with help of fuzzy sets, possibility theory and possibilistic logic. These approaches are suitable for solving ill-defined problems, problems with uncertainties, and partially inconsistent problems. Constraint hierarchies [BFBW92, WB93] define several levels of constraints. The violations of constraints are minimized level by level subsequently with help of weighting the importance of constraints. Partial CSP [FW92] is a general approach which finds a new problem with minimal distance from original problem with help of some metrics. Valued CSP [SFV95] and Semiring-based CSP [BMR97, GC98] (compared in [BFM$^+$96]) define monoid and lattice structures, which specify type of used preferences and manipulation with them. They are very general frameworks for constraint satisfaction and optimization. Above mentioned approaches may be described as a specification of these two frameworks. An approach the closest to ours, the Variable Valued CSP was only briefly described in [LV97, VLS96] and applied for the daily management of an earth observation satellite. There, each variable is associated with some weight expressing its importance. The objective is to produce a partial assignment of the problem variables which satisfies all imperative constraints and maximizes the sum of the weights of the assigned variables.

Preferences in all these systems specify how to compute one or more, best or optimal solution. Unfortunately, the search space is often too large to be fully traversed and explored. In such case some strategy is needed to deal with this situation, however the above mentioned systems don't offer any. The annotations were introduced to help solve such a situation as they suggest some route through the search space which could lead to at least sub-optimal solution.

2.2 Constraint System with Annotations

A constraint satisfaction problem (CSP) is a triple (V, D, C), where

- $V = \{v_1, \ldots, v_n\}$ is the set of variables;
- $D = \{D_1, \ldots, D_n\}$ is the set of domains. Each domain is a finite set containing the possible values for the corresponding variable;
- $C = \{c_1, \ldots, c_n\}$ is the set of constraints. A constraint c_i is a relation defined on a subset $\{v_{i_1}, \ldots, v_{i_{k_i}}\}$ of all the variables, that is $\{D_{i_1} \times \ldots \times D_{i_{k_i}}\} \supseteq c_i$.

Given instantiation θ of the variables, the constraint c_i is satisfied if all the $v_{i_1}, \ldots, v_{i_{k_i}}$ variables got a value such that corresponding value tuple belongs to c_i. A solution of a CSP is such a complete instantiation of the variables that all the constraints are satisfied. If a CSP has not any solution, the problem is called over-constrained or inconsistent and a CSP with more than one solution is called under-constrained.

Over-constrained problems may be solved by putting the over-constrained part of the problem into some objective function which express degree of error in the problem. Often the problem becomes highly under-constrained and large space has to be explored to find the best solution. In many cases different heuristics are used to find some sub-optimal solutions because the whole space can not be explored due to its complexity. This kind of problems and classical optimization problems with partially or even completely ordered variables may become application areas for annotations. Annotations as a source for computing variable ordering may help to find more interesting solutions as soon as possible, especially with help of value ordering heuristics.

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 $\circledast : \mathcal{A}^k \to \mathcal{A}$ 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 $a : C \times V \to \mathcal{A}$ determines the annotation of every variable in every constraint.

Local variable's annotation is used for computing several characteristics of underlying constraint system, 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

$$av : V \to \mathcal{A} \ \ , \ \ av(v) = \underset{\{c \in C \mid v \in var(c)\}}{\circledast} a(c, v) \ \ . \tag{1}$$

Typical examples of $(\mathcal{A}, \preceq, \circledast)$ are intervals (for example $\langle 0, 1 \rangle$) over real numbers, ordering $\geq$ over real numbers, and sum or arithmetic average, resp.

Currently, variable ordering is computed via global variable annotations av which are computed for each variable by combining variable annotations in all constraints where variable occurs (Eqn. 1). During computation variables are instantiated in order given by value of global variable annotation av and ordering of annotations $\preceq$. All variables are instantiated during labeling in order given by importance of variables. This method is combined with selection of values in the domain of variable which are the most interesting — e.g., times between 9 a.m. and 5 p.m. are the most interesting values in the domain of most variables for the lecture time.

3 Timetabling 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 [Sch95, CL98] for surveys). We applied annotations for solving the timetabling problem by constraint programming approach which allows the formulation of all the constraints of the problem in a more declarative way than other approaches [Laj96, GJBP96].

The complexity of timetabling problem often causes that the problem becomes over-constrained. Within the traditional CSP approach, preferences for selection feasible solution are implemented with help of some labelling heuristics [AB94, GKM98]. A formulation of an over-constrained part of the problem may be included in an optimization constraint [FHS95, HW96], other possibility consists in applying weighted [AM98] or hierarchical [BKMQC97] soft constraints. An expensive search of overall solution space may be also replaced by relaxation of difficult constraints either in a semi-automatic [Laj96] or automatic way [GJBP96].

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 course pre-enrollment informations. The similar problem was presented in the paper [GJBP96] which applies constraint logic programming approach together with constraint relaxation. However a number of students was almost ten times smaller than in our case which is really crucial when a solution of NP complete problem has to be find. To our knowledge, any constraint programming system creating automated timetable for individual students or defining similar objective function have not been still described or created for such a large scale problem.

It can be argued that many universities solve similar complex timetabling problem once with just minor modifications for each subsequent year. With this, students are able to plan their courses "years" in advance and are able to visit all interesting or required courses on time. Unfortunately this system is not flexible enough to respect expected degree of freedom during course selection at schools where open credit system with just few strict requirements plays substantial role. Students should be able to create their own timetables wrt. their actual interests, available courses, or current development of research. Our solution tries to respect these requirements and to make timetables flexible enough for everybody.

3.1 Problem Description

Our real 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 required 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 with respect to the number of students pre-enrolled on a course, and with respect to teacher's requirements. Each student should visit one lecture and/or one seminar for each registered course. Computing an ideal timetable should solve the problem of student's splitting into groups such that each student is able to visit lecture and/or seminar for each his/her specified course. Let us define so called *timetable item* representing usual course with its students. Such timetable item is given by a tuple $\langle$course, lecture or seminar identifier/order, set of students$\rangle$. Basically our task consists in instantiation of the set of tuples $\langle$timetable item, class room, time$\rangle$, i.e., each lecture or seminar of course has assigned its set of students, class room, and time[1].

The solution of overall problem should consider the following subproblems:

P_1: splitting students into groups for each course, where an input of this problem are tuples ⟨number of lectures, students for lectures, number of seminars, students for seminars⟩ given for each course, and an output is a set of timetable items defined by tuple ⟨course, lecture or seminar identifier, set of students⟩;

P_2: satisfaction of consistency requirements

- c_1: each teacher gives only one course at a time;
- c_2: only one course can take place in a class room at a time;
- c_3: exception — one teacher can teach one course in two specific class rooms with help of camera and video projection at the same time;

plus required constraints given by teachers

- c_4: requested time of courses wrt. strict unavailability of teacher;
- c_5: time dependencies between some courses, lectures, or seminars;
- ...

P_3: assigning class rooms of suitable size and type (lecture hall, computer room, lecture hall with data projector, ...) to timetable items;

P_4: placing timetable items in time;

P_5: minimization of total number O of timetable items which overlap for any student;

P_6: (partial) satisfaction of preferential constraints (given by teachers or students)

- c_6: Friday afternoon is not preferred;
- c_7: early morning and late evening times generally are not preferred;
- c_8: free time for lunch;
- ...

[1]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.

3.2 Proposed Problem Solution

Generally the subproblems P_1, P_3, and P_4 should cooperate to generate particular sets of students, class rooms, and times such that constraints in P_2 are satisfied, the objective function O in problem P_5 is minimized, and constraints in P_6 are satisfied to the largest possible extent. The complexity of particular subproblems is very high — the standard timetabling problem is NP complete, and so the solution which concurrently combine all these subproblems can not be expected to give results in a reasonable time.

We separated the most complex problem P_1 and we constructed the feasible division of students as an input for other problems. This approach allows us to compute for every two timetable items ti_1, ti_2 a number $n_{ti_1 ti_2}$ of students which need to visit both timetable items. With this we know how desirable or undesirable is parallel teaching of any two timetable items. The objective function O summarize $n_{ti_1 ti_2}$ for any overlapping ti_1, ti_2. This value of O computed for one pre-computed division of students gives us the worst case estimate of generated timetable — some places in lectures/seminars are still free with respect to existing overlapping for some students and students are also allowed to swap lectures/seminars with other students to improve their personal timetables.

The above proposed solution may include requirements in P_2 as standard constraints in constraint satisfaction problem, but enlarged set of constraints including all the requirements in P_6 would make the problem overconstrained. This part of problem may be understood as the second objective which should evaluate "better" placement of particular timetable items measured by "better" satisfaction of preferential constraints. How would be possible to decrease the number of unsatisfiable requirements in such a large-scale problem? This is the point where annotations help us to guide the search through interesting solutions. In our problem, variables important for labeling represent class rooms of timetable items, which may have different size and different equipment (e.g., data projector), and time of timetable items, which are taught by different teachers and pre-enrolled by different number of students. These various degrees of importance are defined through annotations. Value ordering plays important role especially for instantiations of suitable times where each timetable item has specified the more/less preferred times which are assigned first/last.

3.3 Realization

We started our implementation in SICStus Prolog [Int98]. This implementation solved problems P_1, P_4, and partially the problems P_2, P_3 and it was based mainly on global constraints as `cumulative`, `serialized`, and `all_distinct` for sharing discrete and unary resources. This version did not include any specialized search through solution space but was not very efficient — the first solution was found approximately in 1 minute on a PC Pentium 400 MHz. We tried to compare this version with implementation in ILOG [ILO99] software, an object oriented library for constraint programming in C++. Because the first obtained solution was much faster (1 second on the PC Pentium 400 MHz), for the second version which will be described in the following part we used the ILOG environment only.

Splitting of Students (P_1) The output of this problem should be some feasible splitting of students into particular timetable items. Set of registered students 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.

Basic Data Structures in CSP Teachers are represented by unary resources (`IlcUnaryResource` in ILOG), timetable items by non-breakable activities[2] (`IlcIntervalActivity`). Each room of unique size plays a role of unary resource (`IlcUnaryResource`), rooms of the same size are grouped together into one discrete resource (`IlcDiscreteResource`) with capacity equal to the number n of class rooms. 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 class rooms are assigned after solving the CSP. Existence of solution is guaranteed even for non-breakable timetable items [3] and this solution is found in $O(n \times time)$.

[2] Courses are not allowed to be interrupted by definition.

[3] **Proof:** ILOG object `IlcDiscreteResource` representing sets of class rooms of the same size ensures that smaller or equal number of timetabling items than its capacity (the number of class rooms in the set) is allocated in every time. Now let us sort class rooms in any fixed order. Class rooms are assigned to timetable items subsequently. Each class room is assigned to timetable items in increasing time without any gaps if possible. When some timetable items have remained without assigned class rooms in the end, then some gap

Constraints in P_2 Each timetable item `requires` suitable teacher (c_1). Timetable item should require (ILOG constraint `requires`) suitable room (c_2) but this constraint is a bit more complex and exact required resource for each activity/timetable item is described below as a part of resource allocation (P_3). Timetable items taught by one teacher at the same time (c_3) may be easily solved by removing the constraint c_1 for one of the timetable items and binding them to the same time. Specific class rooms are also required by adding new constraints.

Similarly other required constraints are implemented with help of ILOG predefined constraints. If some special constraints on variables are imposed, annotations to variables may be assigned to express that such variable may be a source of propagation and its early instantiation may prune the solution space at the beginning of the search. Typical examples may be strong dependencies between times or class rooms of several timetable items. Let us imagine a simple constraint which binds together times of two timetable items (e.g., part of implementation of the constraint c_3). Higher annotations of these time assignment variables entail their early instantiation when enough space (i.e., free class rooms) is still available for easy time placement of these timetable items. If instantiations of these variables would be postponed, conflicts with other timetable items will be discovered much later and useless parts of the search tree have to be explored.

Resource Allocation (P_3) Each timetable item has to have assigned class room of suitable size and suitable type. Currently we have three different types of class rooms — lecture halls, lecture halls with data projector, and computer rooms — each of them with different sizes. We have created three "hierarchies" of class rooms, the highest level always contains the largest class rooms while the lowest level contains all class rooms. Each timetable item `requires` class room from the sufficient level. Particular levels represent sets of alternative resources (`IlcAltResSet`).

Some type of requirements in resource allocation problem may be violated. Timetable item may be assigned to a slightly smaller class room wrt. the number of pre-enrolled students which means some degree of freedom in the selection of "sufficient" level. Because lecture halls with data projectors are class rooms of the largest size, some timetable items may require lecture hall with data projector even if its number of students is rather

beginning at the starting time of timetable item has to exist wrt. existing discrete resource. But all timetable items were assigned in order given by starting time and any such gap should not exist — the contradiction compelling the proof.

small. Such kind of requirement may by also violated for some timetable items. Set of alternative resources for such timetable items is a union of two levels from different hierarchies representing sufficiently sized lecture halls and all lecture halls with data projector.

Timetable items are assigned to particular class rooms to balance occupation of particular class rooms because even very small degree of freedom allows to find the solution later in the time assignment. Variable ordering during instantiation is given by variables' annotations. Annotations are combined with help of global variable annotation av and sum as a combining function. More important annotations are usually assigned to timetable items with greater number of students or with preferred data projector. Because all annotations of room assignment variables are combined, variable ordering is also influenced by annotations of constraints in P_2 and P_6. Particular variables are instantiated to assign preferred values first and to balance class room occupation. Our variable ordering is given by importance of variables and we are able to assign preferred values in order given by this importance.

Currently results from resource allocation are used as an input for time assignment.

Problems P_4+P_5+P_6 Particular times of timetable items are assigned (P_4) with respect to minimal course overlapping for any students (P_5) and preferential constraints in P_6. Variables are instantiated in order given by their global variable annotation and sum as a combining function. Annotations currently represent preferences of teachers given by the seniority of each teacher and by numbers of students pre-enrolled on the course (expressing interest on a given course).

Each time the variable is instantiated special value ordering is imposed. This value ordering depends on the preferred values in the domain of particular variable v_i and on the number n_i of overlaps with other courses already scheduled at given time. Preferred values are given by preferential constraints in P_6. Usually these constraints prefer times from 9 a.m. to 5 p.m. or restrain Friday's afternoon, but particular value ordering may be specific wrt. teacher's requirement in P_6. The preferred values are selected in increasing order and current value of overlaps n_i is compared with threshold either as an absolute value or as an ratio to number of students in timetable item. If n_i is not sufficient the next remaining value with the highest preferences is selected. The same order is applied during backtracking and searching for better solutions. Annotations ensure that more important variables are assigned first and the possibility of selection of preferred value is increased.

Each solution may be evaluated by sum of overlaps for all timetable items $\sum_{i=1}^{k} n_i$, where k is the number of timetable items. This value gives us exact measure of the solution quality but we do not focus only on the optimality of solution from this point of view and we try to balance this parameter with improved satisfaction of preferential constraints with help of annotations.

Size of Problem 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 units. The number of our courses increases from 220 to 270, each of them is non-breakable and they have variable length from 1 to 4 time units. He have some 100 to 140 teachers, 12–18 class rooms, 1000–1400 students, and from 10 000 to 13 000 student's requirements in course pre-enrollment.

Results Computing the first solution takes from 2 to 4 seconds with respect to the size of problem. Different solutions were obtained for different kinds of threshold (see paragraph Problems P_4+P_5+P_6). The first method applies threshold as a maximal number of students in one timetable items for whom some overlapping with other timetable items may occur. The second method considers threshold as a ratio between the number of students with any overlap and the number of students pre-enrolled to the timetable item. The second method which was slightly better was able to satisfy for all instances 94–95 % of pre-enrollment requirements within the 10–13 % threshold. The first method with 90 to 94 % satisfied requirements allowed maximal number of conflicts between 6 to 11 per timetable item.

Currently achieved results with more than 94 % of satisfied course pre-enrollment requirements in first computed solution promise interesting results even with additional constraints from the problems P_2 and P_6 especially with respect to further improvements in time assignment and resource allocation algorithms. These results seems to be very interesting especially when compared with initial requirements to satisfy 60–70 % course pre-enrollments.

4 Conclusion

Faculty of Informatics timetabling problem allows us to better understand possible semantics of annotations in constraint satisfaction, optimization, or over-constrained problems. The natural basic question "What is the meaning of annotation?" may be answered shortly "It depends (on the point of view).". When too many constraints are introduced and the problem becomes over-constrained, the annotations could help assign more preferred values to more important variables or the annotations could take a role of constraints' weights and subsume constraints ordering [Rud98c]. The annotations in optimization problems may guide the search to find at least sub-optimal solutions through preferred variables and their preferred values, too. Other meaning of variable annotations may be as a pointer of valuable propagation which allows to prune the solution space quickly and leads directly to a solution.

The advantage of our approach may also consists in different definition of the timetabling problem. Usually timetabling problem solved by constraint programming techniques includes large sets of no-clash constraints to teach some sets of courses in different times. Increasing the set of constraints often leads to over-constrained problem and its solution becomes much harder. Our problem definition includes these requirements in the objective function O, which may be manipulated easily than solving directly the over-constrained problem. We have shown that such objective function may be included into a declarative definition of constraint programming problem even for such large scale timetabling problems.

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 also to introduce the annotations as a uniform framework for solving problems with constraints.

We applied annotations for solving real university timetabling problem. We plan to add more constraints, implement more sophisticated resource allocation and time assignment algorithms, e.g., intelligent backtracking or constraining maximal number of fails or maximal consumed time during search of particular branches of the search tree. We would like also to coordinate time assignment algorithm with previous resource allocation process to repair some faulty instantiation by min-conflict repair heuristics.

We also intend to solve other problems from scheduling or planning application areas to show that annotations may help solve wide range of problems.

5 Acknowledgements

This work is supported by the Universities Development Fuund of the Czech Republic under the contract # 0407/1999. We would like to thank Libor Škarvada for the discussions about current creation of the timetable at the Faculty of Informatics.

References

[AB94] Francisco Azevedo and Pedro Manuel Barahona. Timetabling in constraint logic programming. In *Proceedings of the 2nd World Congress on Expert Systems*, January 1994.

[AM98] 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.

[BFBW92] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[BFM+96] Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based CSPs and Valued CSPs: Basic properties and comparison. In Michael Jampel, Eugene Freuder, and Michael Maher, editors, *Over-Constrained Systems*, pages 111–150. Springer-Verlag LNCS 1106, August 1996.

[BKMQC97] Michael Baumgart, Hans Peter Kunz, Sascha Meyer, and Klaus Quibeldey-Cirkel. Priority-driven constraints used for scheduling at univesities. In Mark Wallace, editor, *Practical Application of Constraint Technology*, pages 65–73. The Practical Application Company Ltd, 1997.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, March 1997.

[CL98] 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*, pages 3–19. Springer-Verlag LNCS 1408, 1998.

[DFP96] Didier Dubois, Hélène Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[FHS95] 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.

[FW92] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[GC98] Yan Georget and Philippe Codognet. Compiling Semiring-based constraints with `clp(FD,`S`)`. In Michael Maher and Jean-Franc$_c$ois Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 205–219. Springer-Verlag LNCS 1520, 1998.

[GJBP96] 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.

[GKM98] 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.

[HW96] 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.

[ILO99] ILOG S.A. *ILOG Scheduler 4.4 User's Manual*, 1999.

[Int98] Intelligent Systems Laboratory, Swedish Institute of Computer Science. *SICStus Prolog User's Manual*, 1998.

[Laj96] Gyiri 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.

[LV97] M. Lemaître and G. Verfaillie. Daily management of an earth observation satellite : comparison of ILOG solver with dedicated algorithms for Valued constraint satisfaction problems. In *Proceedings of Third ILOG International Users Meeting*, Paris, France, July 1997.

[Rud98a] Hana Rudová. Constraints with variables' annotations and constraint hierarchies. In Branislav Rovan, editor, *SOFSEM'98: Theory and Practice of Informatics*, pages 409–418. Springer-Verlag LNCS 1521, 1998.

[Rud98b] 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.

[Rud98c] Hana Rudová. Constraints with variables' annotations. Technical Report FIMU-RS-98-04, Faculty of Informatics Masaryk University, 1998. In http://www.fi.muni.cz/informatics/reports/.

[Sch92] Thomas Schiex. Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". In 8^{th} *International Conference on Uncertainty in Artificial Intelligence*, pages 268–275, Stanford, CA, July 1992.

[Sch95] Andrea Schaerf. A survey of automated timetabling. Technical Report CS-R9567, CWI, Amsterdam, NL, 1995.

[SFV95] 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.

[VLS96] G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimization problems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*

(AAAI-96) and Eighth Conference on Innovative Applications of Artificial Intelligence (IAAI-96), pages 181–187, Portland, OR, USA, 1996.

[WB93] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16(3,4):227–318, 1993.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic