On Methods of Constraint-Based Timetabling *

Hans-Joachim Goltz
GMD – German National Research Center for Information Technology
GMD-FIRST, Kekuléstr. 7, D-12489 Berlin, Germany
goltz@first.gmd.de

Abstract

Using Constraint Logic Programming, we develop methods, techniques and concepts for combined interactive and automatic timetabling. An exemples application of our timetabling system proved successful, thus demonstrating the suitability of the methods used. This paper focuses on methods for generating timetables for practical courses. This is a subproblem of our application and can be considered independently of the rest of the problem. The representation of the important constraints is described. Different methods concerning redundant constraints and heuristics for solution search are discussed and computation results are given.

1. Introduction

Constraint Logic Programming over finite domains is a rapidly growing research field aiming at the solution of large combinatorial problems. The Constraint Logic Programming approach has been applied with great success to many real-life problems. A timetabling problem can be suitably modeled in terms of a set of constraints. Various Constraint Logic Programming approaches are discussed, for instance, in [Boizumault et al., 1996, Guéret et al., 1996, Henz and Würtz, 1996, Lajos, 1996, White and Zhang, 1998]. Using Constraint Logic Programming we develop methods, techniques and concepts for combined interactive and automatic timetabling of university courses and school curricula.

An exemple application of such a timetabling system was developed for the Charité Medical Faculty at the Humboldt University, Berlin (see [Goltz and Matzke, 1999]). The Constraint Logic Programming language CHIP (version 5.2) was selected as the implementation language. The global constraints built into CHIP are particularly useful for problem modeling. Complex conditions on sets of variables can be modeled declaratively by such constraints and the efficiency for solution search can be improved.

An essential component of our timetabling system is an automatic heuristic solution search with an interactive user-intervention facility. The user will, however, only be able to alter a timetable or schedule such that no hard constraints are violated. Without this interactive component, our timetabling system could not be used successfully for generating the timetables of the medical faculty. For different reasons, modifications of generated timetables are often required. The timetabling system ensures that no hard constraints are violated by such modifications. Special wishes that are not represented by constraints can often be integrated by modifying an automatically or partially generated timetable. The generated timetables were output as HTML files and are available in the Internet at `http://www.first.gmd.de/plan/charite`.

All components of our timetabling system are implemented in CHIP. This includes also a graphical editor for problem specification and a graphical interface for solution search. About 1.6 Mbytes of program code are used for all these components. Note that the component containing the generation of constraints and the solution search consists of about 190 Kbytes of program code. The initial application of our timetabling system was proved successful, thus demonstrating the suitability of the methods used.

There are different kinds of timetables in our application: schedules for lectures, seminars and practicals. The schedules for lectures and seminars are the same for different weeks. At certain

*in *in Proc. PACLP'2000*, Manchester, April 2000, pp. 167–177

times in their studies, the students have to attend practical courses on different subjects. These courses can only take place within a certain number of weeks. During these weeks, no lectures or seminars are held. Thus, the practical courses can be scheduled independently of the schedules for lectures and for seminars. The generation of timetables for practical courses is considered as a subproblem of our application. This paper focuses on methods for solving this subproblem. Note that this subproblem was not considered in [Goltz and Matzke, 1999].

2. Description of the Problem

Students of medicine have to attend courses covering all medical fields. There are different types of courses: lectures, seminars and practicals. The seminars and practical courses are held in groups of up to 20 students. A timetable consists of different components: schedules for lectures, seminars and practicals. In this paper, we consider only one subproblem: the generation of timetables for practical courses. At certain times in their studies, the students have to attend practical courses on different subjects. Such courses are held in a university clinic or another hospital over a number of days, the course duration depending on the subject being dealt with. A starting day for the practical course must be determined for each group and for each of the given subjects such that the constraints are satisfied. These courses can only take place within a certain number of weeks. During these weeks, no lectures or seminars are held. The practical courses can therefore be scheduled independently. Important constraints for this subproblem are:

C 1: The practicals of each group should not overlap.

C 2: The number of practical courses on the same subject held at any one time is limited.

C 3: In most cases, practical courses on the same subject that are held on any one day have to start on the same day. This means, if P_i and P_j are such courses, either they start on the same day or they do not overlap.

C 4: For practical courses on the same subject, the number of different starting days can be restricted by a maximum number.

C 5: For practical courses on the same subject, the number of different starting days can be restricted by a minimum number.

The other constraints not mentioned here are modeled by simple arithmetic constraints and by the choice of domains. Note that the constraints C 3, C 4 and C 5 are optional constraints and are not required for the practicals of every subject. A solution of the problem is a schedule for the pratical courses that satisfies all given constraints. An automatic relaxation of constraints is not allowed. If a solution is not found, only the user may modify the constraints.

Unlike the schedules for practicals, the schedules for lectures and seminars are the same for different weeks. The lectures and seminars can begin at any time on the quarter hour. Furthermore, there are different kinds of constraints that must be satisfied. For instance, the varying distances between the different locations of the lectures and seminars must be taken into account, and timetabling involves allocating rooms for the individual lectures. The constraints C 3, C 4 and C 5 are not relevant for lectures and seminars. Timetabling for lectures and seminars is not discussed in this paper.

3. Problem Representation

The success of the search often depends directly on the model chosen, and the modeling options on the chosen contraint solver. We use the constraint solver over finite domains of the Constraint Logic Programming language CHIP. The global constraints built into CHIP are particularly useful

for problem modeling ([Beldiceanu and Contejean, 1994]). Global constraints use domain-specific knowledge to obtain better propagation results. Complex conditions on sets of variables can be modeled declaratively by such constraints. Examples of global constraints are `cumulative`, `diffn` and `among`. The `cumulative` constraint was originally introduced to solve scheduling and placement problems. This constraint can be used if the amount of some resource is limited. The `diffn` constraint was included in CHIP to handle multidimensional placement problems. This constraint ensures that a given set of n-dimensional rectangles do not overlap.

The constraints C 1 and C 2 can be easily represented by cumulative constraints. The cumulative constraint ensures that, at each time point in the schedule, the amount of resources consumed does not exceed a given limit L. In our case, the resource needed for course is equal to 1 and the limit L is equal to 1 for C 1. For the constraint C 2, this limit is equal to the maximum number of practical courses on a subject that can be paralleled.

For each practical course P_i, the starting day is modeled by a domain variable $Start_i$, where the domain represents the possible starting days. We consider the starting days of practical courses on the same subject in order to model the constraints C 3, C 4 and C 5. These starting days can be numbered consecutively, and such a number can be assigned to each practical course. A domain variable $StartNr_i$ is introduced for each practical course P_i in order to represent these numbers. For the practicals on a specific subject let n be the maximum number of permitted or possible different starting days. Then, the domain of $StartNr_i$ is defined by $1 \ldots n$.

Assume that $P_1, \ldots, P_m$ are the practical courses on some subject (for m groups), n is defined as above, and let D be the duration (in days) of these courses. Note that the durations of the practical courses on the same subject are not different. Furthermore, let $X_1, \ldots, X_n$ be domain variables with the domains defined by the possible starting days. For each $i, j \in \{1, \ldots, n\}$ with $j = i + 1$ the constraint $X_i < X_j$ is generated. Then, the starting days of the practical courses considered are included in $\{X_1, \ldots, X_n\}$, if for each practical course P_i there is some X_j such that $Start_i = X_j$. This condition can be modeled by the following symbolic constraint:

$$\texttt{element}(StartNr_i, [X_1, \ldots, X_n], [0, \ldots, 0], Start_i, [\texttt{all,all,all,all}])$$

This constraint ensures that the equation $Start_i = X_{StartNr_i}$ is true. The third argument is generally a list $[k_1, \ldots, k_n]$ of natural numbers, and this constraint ensures that the equation $Start_i = X_{StartNr_i} + k_{StartNr_i}$ is true. The last argument is used for controlling propagation. The chosen last argument means that the constraint is always woken if the domain of some variable belonging to the constraint is changed.

If the number of different starting days is restricted by a maximum number (constraint C 4), then this constraint can be modeled by the choice of the permitted number of starting days n.

Let Max be the number of days on which a practical course can be held. If we assume that the practical courses $P_1, \ldots, P_m$ have to satisfy the constraint C 3, then the maximum number n of permitted different starting days is restricted by Max/D. The constraint $X_i + D \leq X_j$ is generated for each $i, j \in \{1, \ldots, n\}$ with $j = i+1$. From these constraints and the relations defined above, it follows that for each P_i and P_j one of the following conditions holds: $Start_i = Start_j$, $Start_i + D \leq Start_j$, or $Start_j + D \leq Start_i$. Thus, the constaint C 3 is satisfied.

Now we assume that $P_1, \ldots, P_m$ have to satisfy the constaint C 5. This means that the number of different starting days is restricted by a minimum number Min. Let $MaxPar$ be the maximum number of practical courses that can be parallelled and let $N_1, \ldots, N_n$ be domain variables with the domains $0 \ldots MaxPar$. Such a domain variable N_k will represent the number of elements belonging to the following subset of the practicals: $\{ P_i \mid Start_i = X_k, \ 1 \leq i \leq m \}$. Note that $Start_i = X_k$ if $StartNr_i = k$. Obviously, the domain variables $N_1, \ldots, N_n$ have to satisfy the equation $N_1 + \ldots + N_n = m$. The definition of the variables $N_1, \ldots, N_n$ can be modeled by `among` constraints. For each $k \in \{1, \ldots, n\}$, the following constraint is generated[†]:

[†]In CHIP, this set of `among` constraints can be integrated into one special `among` constraint.

$$\texttt{among}(N_k, [StartNr_1, \ldots, StartNr_m], [0, \ldots, 0], [k], \texttt{all})$$

Such a constraint is satisfied if exactly N_k-many variables of $\{StartNr_1, \ldots, StartNr_m\}$ are assigned to the value k. The third argument is generally a list of natural numbers with m elements. A difference from the value k can be expressed by this argument. The last argument is used for controlling propagation.

For each subject, the practical courses have to satisfy these **among** constraints and the equation with respect to the sum of $N_1, \ldots, N_n$. In the next section, we regard these constraints as redundant constraints. In order to satisfy constraint C 5, a further condition has to be fulfilled: only a determined number of the variables $\{N_1, \ldots, N_n\}$ can take at most the value 0. This can be modeled by a further **among** constraint, where Z is a domain variable with the domain $0 \ldots (n - Min)$:

$$\texttt{among}(Z, [N_1, \ldots, N_n], [0, \ldots, 0], [0], \texttt{all})$$

This constraint is satisfied if at most $(n - Min)$-many variables of $N_1, \ldots, N_n$ take the value 0.

4. Redundant Constraints

Redundant constraints are additional constraints followed logically from the other constraints. Such constraints can improve the propagation properties. Thus, the search space can be reduced. However, there is no gurantee of this and the computation time for a search step can be increased if redundant constraints are added. We therefore investigated different possibilities of adding redundant constraints.

The constraints C 1 and C 2 can also be modeled by **diffn** constraints. For example, let $Start_1, \ldots, Start_m$ be the domain variables of starting days of the practical courses on the same subject, let D be the duration of these courses, and let $MaxPar$ be the maximum number of courses that can be given at any one time. For each of these practical courses, we add a domain variable Par_i with the domain $1 \ldots MaxPar$. We consider a practical course as a "2-dimensional rectangle" with the dimensions "*time*" and "*parallelism*". Then, the constraint C 2 is satisfied if the corresponding rectangles do not overlap. This can be modeled by

$$\texttt{diffn}([Start_1, D, Par_1, 1], \ldots, [Start_m, D, Par_m, 1]).$$

This constraint can also be modeled by using the domain variables $StartNr_i$:

$$\texttt{diffn}([StartNr_1, 1, Par_1, 1], \ldots, [StartNr_m, 1, Par_m, 1]).$$

If C 1 is represented in this way, then Par_i is equal to 1. In the sequel we regard these **diffn**-constraints as redundant.

The **among** constraints related to the variables $StartNr_1, \ldots, StartNr_m$ and described in Section 3 can be generated for the practical courses on each subject. As mentioned above, these constraints are redundant if they are not required for constraint C 5. Such **among** constraints can also be generated with respect to the domain variables $Start_1, \ldots, Start_m$. In this case, a variable N_i must be defined for each possible starting day. Note that the equation $N_1 + \ldots + N_n = m$ is also generated if the corresponding **among** constraints are generated for the practical courses of a subject.

For any two practical courses P_i und P_j on some subject, the following conditional constraints concerning the relations between both kinds of domain variables can be added:

$$\texttt{if}\ \ Start_i < Start_j\ \ \texttt{then}\ \ StartNr_i < StartNr_j\ \ \texttt{else}\ \ StartNr_i \geq StartNr_j$$
$$\texttt{if}\ \ StartNr_i < StartNr_j\ \ \texttt{then}\ \ Start_i < Start_j\ \ \texttt{else}\ \ Start_i \geq Start_j$$

If such conditional constraints are added for all relations of this kind, then, generally, the number of these constraints is relatively large.

5. Solution Search

A constraint solver over finite domains is not complete because consistency is only proved locally. Thus, a search is generally necessary to find a solution. Often, this search is called "labeling". The basic idea behind this procedure is: select a variable from the set of problem variables considered, choose a value from the domain of this variable and then assign this value to the variable; backtracking occurs if the constraint solver detects a contradiction; repeat this until all problem variables have a value and the constraints are satisfied. This search includes two kinds of nondeterminism: *selection of a domain variable* and *choice of a value* concerning the selected variable. Methods for value choice are not considered in this paper. It is well known that the heuristic used for variable selection exerts a great influence on the solution search.

We use a static ordering for variable selection based on an ordering for the practical courses. There are two basic variants for sorting the practical courses: *sorting with respect to groups* (first all courses of one group, then all courses of another group, and so on) and *sorting with respect to subjects* (first all courses of one subject, then all courses of another subject, and so on). Our experience has shown that the second variant is better. We therefore suppose that each subject has a priority and that the ordering of the practical courses is determined by these priorities, the second parameter for the ordering being the serial number of the group.

For a given ordering of practical courses, we consider three different strategies for assigning values to the variables $Start_i$ and $StartNr_i$ concerning each course P_i:

S 1: Values are assigned step by step to the variables $Start_i$ **and** $StartNr_i$ for each practical course P_i.

S 2: At first, for all practical courses, values are assigned step by step to the variables $StartNr_1$, $StartNr_2$, ... Then, values are assigned step by step to the variables $Start_1$, $Start_2$, ...

S 3: First, for all practical courses, values are assigned step by step to the variables $Start_1$, $Start_2$, ... Then, values are assigned step by step to the variables $StartNr_1$, $StartNr_2$, ...

The generation of a timetable can also be controlled interactively by the user. The following interactive timetabling actions are possible with the help of the graphical interface: scheduling an individual course, scheduling marked courses automatically, removing marked courses from the timetable, moving an individual course within the timetable, and scheduling the remaining courses automatically. These actions can be perfomed in any order or combination, and no automatic backtracking is caused by such user actions. The user can, however, only alter a timetable in such as way that no hard constraints are violated. This interactive component of our timetabling system is very important. It makes step by step generation of a timetable possible. In some cases, such generation was necessary to find a solution. Modifications of generated timetables are often required for different reasons. Special wishes that are not represented by constraints can often be integrated by modifying an automatically or partially generated timetable. The timetabling system ensures that no hard constraints are violated by such modifications.

In the sequel, we consider automatic search only. For automatic scheduling of a certain set of courses, we use the same methods as for the automatic generation of the whole timetable.

6. Computation Results

Some experimental results are presented in this section. We consider two practical instances of our timetabling problems (summer semester 1999 and winter semester 1999/2000). In both cases practical courses on ten different subjects are scheduled. 14 groups and 14 weeks are considered in the first case, and 20 groups and 16 weeks in the second. Figures 1 and 2 show two solutions to these problems. Note that these solutions are interactively modified versions of solutions that were generated automatically. The solutions of these problems are also available in

the Internet at `http://www.first.gmd.de/plan/~goltz/k9b5.gif` (see also `.../w9b_5b.htm`) and `http://www.first.gmd.de/plan/~goltz/w9b5.gif` (see also `.../k9s_5b.htm`), respectively.

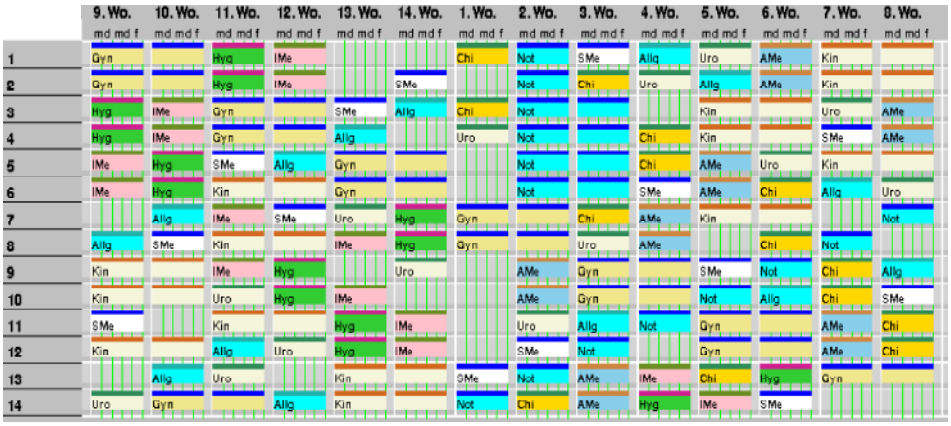

Figure 1: A solution for the examples *Ex 11*, *Ex 12*, *Ex 13*

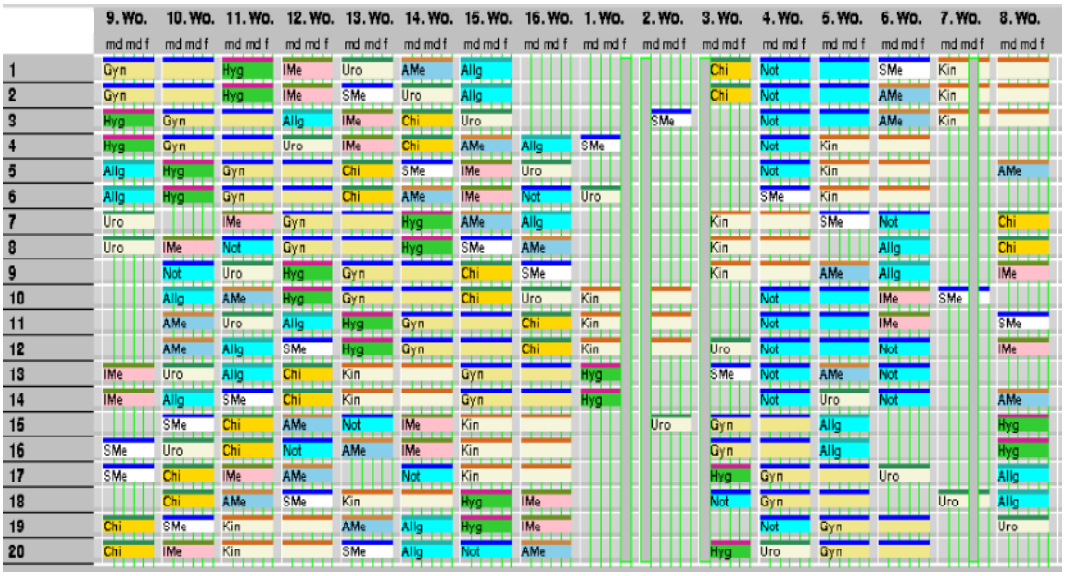

Figure 2: A solution for the examples *Ex 21*, *Ex 22*

The examples *Ex 11*, *Ex 12* and *Ex 13* are different representations of the first case and the examples *Ex 21* and *Ex 22* different representations of the second. The constraints C 1 and C 2 are required in each example (for each group and each subject). For the practicals on one subject, a specific week is preferred. The constraint C 3 is used for the practicals on six subjects relating to the first case and for the practicals on seven subjects relating to the second. For the same practicals, the constraint C 4 is required. The constraint C 5 is only used for the practicals on two subjects relating to the examples *Ex 12*, *Ex 13* and *Ex 22*.

If the practical courses on some specific subject have to be held for all groups, then these courses can also be decomposed into two disjunct sets. The examples *Ex 11* and *Ex 12* differ in term of this kind of modeling and the priorities of the subjects. The only difference between the

examples *Ex 12* and *Ex 13* is the priority of **one** subject. As regards the second case, the following condition must be satisfied for the practical courses on two subjects, where the duration of these courses is one week: for each of the two subjects and for each week, at least one practical course must be held. This condition can be expressed by constraint C 5 and is used in example *Ex 22* . Another way of modeling is used in example *Ex 21* . The practical courses on such a subject are decomposed into two disjunct sets: one for 16 groups and one for the other 4 groups. Furthermore, we take into account the constraint that the practical courses of such a set should not overlap. The examples *Ex 21* and *Ex 22* do not differ with respect to the priorities of the subjects. Thus, the same ordering of practical courses is used for solution search in both of these examples.

The following posibilities for adding redundant constraints are considered in this paper (see Section 4 as well):

diffn: for the constraints C 1 and C 2 , additional `diffn` constraints concerning both kinds of domain variables

among-start: additional `among` constraints concerning the domain variables $Start_i$

among-nr: additional `among` constraints concerning the domain variables $StartNr_i$

if: conditional constraints for all relations between both kinds of domain variables

These possibilities are combined into 7 different methods for adding redundant constraints. These methods are defined in the Table 1. The sign "+" means that these kinds of redundant constraints are added, and the sign "−" means that they are not added.

	among			
	start	*nr*	*diffn*	*if*
a	−	−	−	−
b	−	−	+	+
c	+	−	+	+
d	−	+	+	+
e	−	+	+	−
f	−	+	−	+
g	−	+	−	−

Table 1: Definition of methods

Table 2 presents several computation results for several methods and the five examples. We use the three strategies S 1 , S 2 and S 3 defined above and 7 different methods (a, b, c, d, e, g, f) for adding redundant constraints. Note that our experiments also included other strategies and other combinations for adding redundant constraints. Only the interesting results are presented in this paper.

For the search, the number of permitted search steps (backtracking steps) was restricted to 1,000. The figures in the columns with the sign "#" show the number of backtracking steps needed for the search. The sign "–" means that no solution was found within the permitted search steps. All execution times given are for a *Sun ULTRA 1* (in seconds) and relate to the time needed for the search.

Interesting conclusions for the computation results given in Table 2 are:

- Within { S 1 , S 2 , S 3 } , there is no best search strategy. If only one of these strategies is selected, then no solution is found for *Ex 12* or *Ex 22*.
- The computation results of the search strategies S 1 and S 2 are relatively similar. In most cases, a solution was found (within the permitted backtracking steps) with strategy S 1 if and only if a solution was found with strategy S 2.

method		*Ex 11*		*Ex 12*		*Ex 13*		*Ex 21*		*Ex 22*	
		#	sec	#	sec	#	sec	#	sec	#	sec
	a	–	4.7	–	4.7	–	4.3	–	4.6	778	5.1
	b	–	14.5	–	15.7	–	23.9	–	11.5	667	12.5
	c	11	1.1	–	30.9	83	3.0	24	2.1	184	5.9
S 1	d	0	0.8	–	37.1	31	1.8	0	1.6	1	1.6
	e	1	0.6	–	28.9	39	1.6	0	1.3	2	1.4
	f	482	9.6	–	23.4	814	10.3	0	0.9	1	1.0
	g	212	2.6	–	13.8	328	2.6	0	0.6	2	0.6
	a	–	6.4	–	4.3	–	4.4	–	6.3	138	1.2
	b	–	28.4	–	12.0	–	10.5	–	11.4	69	2.6
	c	–	29.9	–	15.6	–	13.2	33	2.6	69	2.8
S 2	d	0	0.8	–	34.0	14	0.9	0	1.6	0	1.7
	e	0	0.7	–	28.9	14	0.8	0	1.4	0	1.4
	f	259	4.4	–	9.6	242	3.0	0	0.8	0	0.8
	g	259	3.3	–	8.0	246	2.2	0	0.7	0	0.6
	a	84	0.5	–	5.6	–	4.0	–	4.7	–	8.4
	b	2	0.6	186	3.3	–	27.8	–	11.5	–	26.2
	c	1	0.6	11	0.8	44	2.0	0	1.4	–	54.1
S 3	d	1	0.7	10	0.9	23	1.4	0	1.4	–	52.2
	e	25	0.8	34	1.1	36	1.4	–	20.7	–	20.7
	f	1	0.3	–	13.6	84	1.3	0	0.7	–	25.8
	g	25	0.4	–	7.4	84	0.9	–	9.6	–	10.1

Table 2: Comparison of several methods

- The best method for adding redundant constraints is method "d" .
- Although the computation time can be increased if redundant constraints are added, the generation of redundant constraints is generally advantageous. Without redundant constraints no solutions were found for the examples *Ex 12*, *Ex 13* and *Ex 21*.
- Additional `among` constraints should be generated with respect to the domain variables $StartNr_i$ since the results of method "d" are better than the results of method "c" .
- In some cases, redundant constraints do not reduce the search space. For instance, if strategy S 1 or S 2 is used, then the best method is "g" for *Ex 21* and *Ex 22*, and additional `if` constraints do not reduce the search space.
- In order to generate the constraints for all relations between both kinds of domain variables, 1,708 conditional constraints are generated for each of the examples *Ex 11*, *Ex 12* and *Ex 13*. 3,268 constraints of this kind are neeeded for each of the other examples. Although the number of these redundant constraints is relatively large, the additional time needed for the search is comparatively insignificant.
- The examples *Ex 12* and *Ex 13* show how a small change in the variable ordering can affect the computation results.

7. Conclusion

Our experience has shown that in many cases either a solution can be found within only a few backtracking steps, or a large number of backtracking steps are needed. This is also confirmed by the computation results presented in this paper. We therefore use the following basic search method: *the number of backtracking steps is restricted, and different heuristics are tried out*. This means, that backtracking is carried out on different heuristics. The computation results also show

that the search space can be reduced considerably by adding redundant constraints. These results are also useful for other problems involving limited resources.

With regard to the problem discussed in this paper, the user can choose between different methods for the solution search. In particular, the user can control the following parameters: the number of attempts with different heuristics, the number of permitted backtracking steps for one attempt, the strategy for assigning values to both kinds of domain variables, the priorities of the subjects, and which redundant constraints should be added.

For all problems of our application a solution was be able to be generated. In most cases an initial solution for a timetable was found within a few seconds. In other cases, an initial solution was found within ten minutes. If an initial solution could not be found within a short time, then a solution was found by an interactive step-by-step search within a few minutes. For different reasons, an initial solution was generally modified interactively.

Our future work on timetabling problems will include investigating better heuristics for variable selection. The methods, techniques and concepts already developed or under development will also be tested on other timetabling applications.

References

S. Abdennadher and M. Marte (1998). University timetabling using constraint handling rules. In O. Ridoux, editor, *Proc. JFPLC'98, Journées Francophones de Programmation Logique et Programmation par Constraintes*, pages 39–49, Paris, Hermes.

Beldiceanu, E. and Contejean, E. (1994). Introducing global constraints in CHIP.
J. Mathematical and Computer Modelling, 20(12):97–123.

Boizumault, P., Delon, Y., and Peridy, L. (1996). Constraint logic programming for examination timetabling. *J. Logic Programming*, 26(2):217–233.

Burke, E. and Carter, M., editors (1998). *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg.

Burke, E. and Ross, P., editors (1996). *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg.

T. B. Cooper and J. H. Kingston (1996). The complexity of timetable construction problems. In *[Burke and Ross, 1996]*, pages 183–295.

Goltz, H.-J. and Matzke, D. (1999). University timetabling using constraint logic programming. In *Proceedings PACLP'99*, pages 529–535. The Practical Application Company Ltd.

Guéret, C., Jussien, N., Boizumault, P., and Prins, C. (1996). Building university timetables using constraint logic programming. In *[Burke and Ross, 1996]*, pages 130–145.

Henz, M. and Würtz, J. (1996). Using Oz for college timetabling.
In *[Burke and Ross, 1996]*, pages 162–177.

Lajos, G. (1996). Complete university modular timetabling using constraint logic programming. In *[Burke and Ross, 1996]*, pages 146–161.

Marriott, K. and Stucky, P. J. (1998). *Programming with Constraints: An Introduction*. The MIT Press, Cambridge (MA), London.

A. Schaerf (1995). A survey of automated timetabling. Technical Report CS-R9567, Centrum voor Wiskunde en Informatica.

White, G. and Zhang, J. (1998). Generating complete university timetables by combining tabu search with constraint logic. In *[Burke and Carter, 1998]*, pages 187–198.