Combined Automatic and Interactive Timetabling Using Constraint Logic Programming *

Hans-Joachim Goltz

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

Abstract. We use Constraint Logic Programming and develop methods, techniques and concepts for a combined automatic and interactive timetabling of university courses and school curricula. An instance of such a timetabling system was developed for the Charité Medical Faculty at the Humboldt University, Berlin. This paper looks at methods and techniques for solving this timetabling problem. An essential component 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. Initial application of our timetabling system has proved successful and demonstrated the suitability of the methods used. The results obtained are also useful for solving other problems.

1 Introduction

A timetabling problem can be defined as the scheduling of a certain number of courses that have to satisfy specific conditions, in particular constraints regarding demands on limited resources. Timetabling has long been known to belong to the class of problems called NP-complete (see, e.g., [7]). Different software methods and approaches are available for automated timetabling (see, e.g., [18]). Constraint Logic Programming over finite domains is a rapidly growing research field aimed 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 modelled in terms of a set of constraints. Various Constraint Logic Programming approaches are discussed, e.g., in [1, 2, 4, 12, 14, 16, 15, 19].

In [2], the Constraint Logic Programming language DOMLOG is used to solve the timetabling problem of a computer science department. This language contains user-defined heuristics for search methods and allows greater flexibility in the declaration of "forward" and "look-ahead" constraints (compared with other such languages). The timetabling problems of a computer science department

* in: E. Burke and W. Erben (eds.), Proc. PATAT 2000, Konstanz, August 2000, pp. 78 – 95

are also addressed in [15] and [1], the former using the Constraint Logic Programming language ECL^{i}PS^e and the latter choosing the high-level constraint language Constraint Handling Rules (CHR) as the implementation language. The language CHR was developed by Tom Frühwirth [8]. Based on an existing domain solver written in CHR, a solver for the timetabling problem was developed, requiring as little as 20 lines of code. [4] and [12] look at the application of the Constraint Logic Programming language CHIP to a university's examination timetabling problems and describe the use of the global constraint *cumulative*. Different labelling strategies are discussed in [12].

M. Henz and J. Würtz [14] present a solution for the timetabling problem of a German college using the language Oz. This language was developed by the DFKI at Saarbrücken (Germany) and is a concurrent language allowing functional, object-oriented and constraint programming. G. Lajos [16] describes experience in constructing a large-scale modular timetable using Constraint Logic Programming. Arithmetic constraints and the built-in constraint *atmost* are only used for problem modelling. In [19], a hybrid approach for the automatic generation of university and college timetables is presented. This approach combines features of constraint logic and tabu search. These two approaches are also discussed separately.

Our research is concerned with the development of methods, techniques and concepts for a combined automatic and interactive timetabling of university courses and school curricula. The timetabling systems must be flexible enough to take into account special user requirements. An essential component 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. Our research focuses on the following areas: concepts and methods for modelling timetabling problems; examination of the influence of different modelling techniques on the solution search; development and comparative analysis of different methods and techniques for heuristic solution search; interactive user intervention in the solution search via a graphical interface. The methods, techniques and concepts developed or under development are tested and further elaborated on prototypical applications.

An instance of such a timetabling system was developed for the Charité Medical Faculty at the Humboldt University, Berlin. The Constraint Logic Programming language CHIP (version 5.2) was selected as the implementation language. The global constraints and the object-oriented components built into CHIP are particularly useful for problem modelling. The generated timetables were output as HTML files and are available in the Internet. This initial application of our timetabling system has proved successful and has demonstrated the suitability of the methods used. The interactive component of our timetabling system is very important. Without this component, it could not be used successfully for generating the medical faculty's timetables.

This paper contains a brief discussion of this application and the methods developed to solve the timetabling problems of university courses. The experience gained with this application can be used for other applications as well. This

paper draws together elements from [10, 11] and presents them in revised and extended form. Section 2 gives a brief description of the timetabling problem. This is followed by a discussion of problem modelling. Some aspects of solution search are treated in Section 4. The next section describes interactive timetabling actions. Some remarks on the implementation are included and examples of experimental results are given.

2 The Timetabling 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 practicals are held in groups of up to 20 students. Each student in a particular semester participates in one of these groups, there being between 10 and 24 groups per semester. The lectures and seminars may begin on the quarter hour and may run for different lengths of time. The buildings of the Humboldt University's Charité Medical Faculty are at two different locations in Berlin. Some courses are also held in clinics and hospitals in other parts of Berlin. Lectures on the same subject are given in parallel in different parts of the faculty. Thus a student can choose which of the parallel lectures to attend.

At certain stages 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. Unlike the schedules for practicals, the schedules for lectures and seminars are the same week for week.

Other important constraints for this timetabling problem are:

C 1: There are restrictions with respect to the starting time for each course.
C 2: Two lectures on the same subject may not be scheduled for the same day.
C 3: The lectures, seminars and practicals of each group should not overlap.
C 4: The number of courses on the same subject held at any one time is limited.
C 5: Some seminars are only held during certain weeks.
C 6: Timetabling also involves allocating rooms for the individual lectures.
C 7: Some lectures can only be held in certain specified rooms; in some cases only one room matches the requirements.
C 8: The varying distances between the different course locations must be taken into account.
C 9: In most cases, practical courses on the same subject must begin on the same day or may not overlap. This means, if P_i and P_j are such courses, they either start on the same day or they are not allowed to overlap.
C 10: For practical courses on the same subject, the number of different starting days can be restricted by a maximum number.

C 11: For practical courses on the same subject, the number of different starting days can be restricted by a minimum number.

C 12: The starting time preferences for courses and the preferred rooms for lectures should also be taken into account where possible.

The last constraint is a soft constraint. The other constraints are hard constraints and must be satisfied. Other constraints not mentioned here are modelled by simple arithmetic constraints and by the choice of domains. Note that the constraints C 9, C 10 and C 11 are optional constraints and are not required for the practicals of every subject.

A solution of the problem is offered by schedules for the different kinds of courses that satisfy all given hard constraints. An automatic relaxation of hard constraints is not allowed. If no solution is found, only the user is allowed to modify the hard constraints.

3 Problem Modelling

3.1 Constraint Logic Programming

Constraint Logic Programming (CLP) is a generalization of Logic Programming, unification being replaced by constraint handling in a constraint system. CLP combines the declarativity of Logic Programming with the efficiency of constraint-solving algorithms. Constraint Logic Programming with constraints over finite integer domains, CLP(FD), has been established as a practical tool for solving discrete combinatorial problems. Each constrained variable has a domain that must first be defined. Such a variable is also called a domain variable. The usual arithmetic constraints *equality*, *disequality* and *inequalities* can be applied over linear terms based on domain variables and natural numbers.

A timetabling problem can be suitably modelled in terms of a set of constraints. The method chosen for problem modelling has a considerable influence on the solution search. The success of the search often depends directly on the model chosen, and the modelling options depend 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 modelling ([3]). Global constraints use domain-specific knowledge to obtain better propagation results. Complex conditions on sets of variables can be modelled declaratively by such constraints and can be used in multiple contexts. Global constraints with specialized consistency methods can greatly improve efficiency for solving real-life problems. Examples of global constraints are `cumulative`, `diffn` and `among`.

The `cumulative` constraint was originally introduced to solve scheduling and placement problems. This constraint ensures that, at each time point in the schedule, the amount of resources consumed does not exceed the given limit. The `diffn` constraint was included in CHIP to handle multidimensional placement problems. The basic `diffn` constraint takes as one argument a list of n-dimensional rectangles, where origin and length can be domain variables with respect

to each dimension. This constraint ensures that a given set of n-dimensional rectangles do not overlap. Special conditions of n-dimensional rectangles can also be expressed by `diffn` constraints. The consistency methods for these constraints are very complex because there are numerous possibilities for inferring new information, depending on which of the variables are known. The `among` constraint was introduced in order to specify the way values can be assigned to variables. There are different variants of this constraint.

A useful symbolic constraint is `element(N,List,Value)`. It specifies that the $\mathtt{N}^{th}$ element of the nonempty list `List` of natural numbers must have the value `Value`, where `Value` is either a natural number, a domain variable or a free variable. There are also other variants of this constraint in CHIP.

3.2 Lectures and seminars

This section considers the representation of the conditions for lectures and seminars. For each lecture and seminar, the *starting time* is represented by a domain variable. If we assume that such a course can be held on five days of a certain week between 8 a.m. and 8 p.m., then $5 \times 12 \times 4 = 240$ time units have to be considered. This means, initially, that the domains for the starting time variables are defined by the natural numbers $1, 2, \ldots, 240$. The attributes *room*, *week* and *location* of a course may be domain variables if these values are unknown. The possible values of these attributes are mapped into the natural numbers. The choice of the domains includes restrictions of the values (constraints C 1 and C 7). Note that the capacity of a room is taken into account by the choice of the domain of a room variable. With the exception of the last constraint, all other constraints of the timetabling problem are directly represented by built-in constraints. The last constraint C 12 is a soft constraint and is integrated into the solution search.

If two lectures are not scheduled for the same day (C 2), then one lecture is scheduled at least one day before the other lecture. Since two lectures on the same subject have the same features, we can determine which lecture is scheduled at least one day before the other. For example, assume that S_1 `in` 1..240 and S_2 `in` 1..240 are the domain variables for the starting times of two such lectures. Furthermore, let X `in` $[48, 96, 144, 192]$ be a new domain variable, where $\{48, 96, 144, 192\}$ are the last time units of the first four days. Then, the relations $S_1 < X$ and $X < S_2$ ensure the constraint C 2. This method of modelling is more efficient than the method suggested in [12] for the same constraint. However, our method can only be used if the two lectures have the same features. The method given in [12] can also be used in other cases.

The constraints C 3 (*non-overlapping of a set of courses*) and C 4 (*limited resources for a set of courses*) can be represented by `cumulative` constraints. For example, let $S_1, S_2, \ldots, S_n$ be the domain variables of the starting times of a set of courses and let $D_1, D_2, \ldots, D_n$ be the corresponding lengths of time. If Max is the maximum number of courses of this set that can be given at any

one time, then this constraint can be modelled by[1]:

$$\texttt{cumulative}([S_1, S_2, \ldots, S_n], [D_1, D_2, \ldots, D_n], [1, 1, \ldots, 1], Max)$$

If a set of courses should not overlap, then such a constraint can be modelled analogously, Max being equal to 1 in this case.

If starting times and rooms have to be determined for a set of courses (constraint C 6), these courses can be considered as a "two-dimensional rectangle" with the dimensions "*time*" and "*room*", and these rectangles may not overlap. The use of `diffn` constraints can ensure that this constraint is satisfied. For example, let $S_1, S_2, \ldots, S_n$ be the domain variables of the starting times, $D_1, D_2, \ldots, D_n$ the corresponding durations, and let $R_1, R_2, \ldots, R_n$ be the domain variables for room allocations. A "two-dimensional rectangle" is then represented by $[S_i, D_i, R_i, 1]$, where the length of the dimension *"room"* is 1 unit. The constraint

$$\texttt{diffn}([[S_1, D_1, R_1, 1], [S_2, D_2, R_2, 1], \ldots, [S_n, D_n, R_n, 1]])$$

ensures that these rectangles do not overlap, i.e., at any one time, at most one course is assigned to each room. This constraint can also be modelled by the cumulative constraint, where additional domain variables $X_1, X_2, \ldots, X_n$ are required. For each i, the variables S_i, R_i, X_i are linked by the equation $X_i = N * R_i + S_i$, where N is greater than or equal to the maximum number of time units. A similar method of modelling with cumulative constraints is discussed in [4]. However, the domains of the variables are more reduced if the conditions are defined using the `diffn` constraint.

The `diffn` constraint can also be used for modelling conditions C 5 (*some seminars are only held during certain weeks*) and C 8 (*the varying distances between the different course locations must be taken into account*). At least the dimensions "*time*" and "*location*" are used for modelling condition C 8. Breaks are needed between courses. The length of a break depends on the distance between locations. These breaks are considered as dummy courses. Such a dummy course is defined for each course and each location. The starting time of a dummy course has to be equal to the finishing time of the corresponding course. The duration of a dummy course is defined by the symbolic built-in constraint `element/3` and depends on the selected location of the corresponding course. If this modelling method is used, then a required break should not be longer than the duration of a relevant course. This requirement is satisfied for our application.

3.3 Practicals

This section looks at the modelling of constraints for practicals. For each practical course P_i, the starting day is modelled by a domain variable $Start_i$, where the domain represents the possible starting days. Note that restrictions with respect to the starting days (constraint C 1) is taken into account by the domain choice. We consider the starting days of practical courses on the same subject in

[1] The unused arguments are not mentioned.

order to model the constraints C 9 , C 10 and C 11 . These starting days can be numbered consecutively and a corresponding number assigned to each practical course. A domain variable $StartNr_i$ is introduced for each practical course P_i 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 $1 \ldots n$.

In the same way as for lectures and seminars, the constraints C 3 (*non-overlapping of a set of practicals*) and C 4 (*limited resources for a set of practicals*) can easily be represented by `cumulative` constraints. We generate `cumulative` constraints for both kinds of domain variables.

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 modelled 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, this constraint ensuring 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 10), then this constraint can be modelled 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 9 , 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 9 is satisfied.

Now we assume that $P_1, \ldots, P_m$ have to satisfy the constaint C 11 . 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 held in parallel 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 modelled

by `among` constraints. For each $k \in \{1, \ldots, n\}$, the following constraint is generated[2]:

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

Such an `among` constraint is satisfied if exactly N_k-many variables of the set $\{StartNr_1, \ldots, StartNr_m\}$ are assigned to the value k. The third argument is generally a list of natural numbers with m elements. A deviation 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. In order to satisfy constraint C 11, a further condition has to be fulfilled: only a determined number of the variables $\{N_1, \ldots, N_n\}$ can take on at most the value 0. This can also be modelled by an `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 on the value 0.

3.4 Redundant constraints

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

The constraints C 3 (*non-overlapping of a set of practicals*) and C 4 (*limited resources for a set of practicals*) can be also modelled by `diffn` constraints.

For example, let $Start_1, \ldots, Start_m$ be the domain variables of the 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 view a practical course as a "two-dimensional rectangle" with the dimensions "*time*" and "*parallelism*". Then, the constraint C 4 can be modelled by

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

If C 3 is represented in this way, then Par_i is equal to 1. We regard these `diffn` constraints as redundant and assume that the two kinds of cumulative constraints are generated in each case.

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

The `among` constraints described in Section 3.3 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 11 . Such `among` constraints can also be generated for 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 and P_j on some subject, the following conditional constraints concerning the relations between the two kinds of domain variables can be added:

`if` $Start_i < Start_j$ `then` $StartNr_i < StartNr_j$ `else` $StartNr_i \geq StartNr_j$
`if` $StartNr_i < StartNr_j$ `then` $Start_i < Start_j$ `else` $Start_i \geq Start_j$

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

4 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 "labelling". The basic idea behind this procedure is to 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 must be used if the constraint solver detects a contradiction; repeat this until all problem variables have a value and the constraints are satisfied. In our timetabling system, the domain-reducing strategy is also used for the search. This strategy is a generalization of the labelling method and was presented in [9]:

- The assignment of a value to the selected variable is replaced by reduction of the domain of this variable.
- If backtracking occurs, the unused part of the domain is taken as the new domain for repeated application of this method.

A reduced domain should be neither too small nor too large. A solution is narrowed down by this reduction procedure, but it does not normally generate a solution to the problem. Thus, after domain reduction, assignment of values to the variables must be performed, which may also include a search. The main part of the search, however, is carried out by the domain-reducing procedure. A conventional labelling algorithm can be used for the final value assignment. If a contradiction is detected during final value assignment, the search can backtrack into the reducing procedure.

The advantages of this method have been shown by many examples. However, there are also problems where the domain-reducing strategy offers no benefits. For instance, this method has neither advantages nor disadvantages for scheduling the practical courses. For scheduling the lectures and seminars the advantages

of this method have been shown by our experiments. In particular, the different breaks needed between courses can be better integrated into the solution search.

The search includes two kinds of nondeterminism: *selection of a domain variable* and *choice of a reduced domain* concerning the selected variable. If labelling is used, the reduced domain is one value. It is well known that the heuristic used for variable selection exerts a considerable influence on the solution search. We use a static ordering for variable selection based on an ordering for the courses. There are two basic variants for sorting the seminars and the practical courses: *sorting by group* (first all the courses of one group, then all the courses of another group, and so on) and *sorting by subject* (first all the courses of one subject, then all the courses of another subject, and so on). Our experience has shown that the second variant is better. We therefore assume that each subject has a priority and that the ordering of the seminars and the practical courses is determined by these priorities, the second parameter for the ordering being the serial number of the group. The lectures are also ordered by the priorities of the subjects.

The chosen heuristics for determining the reduced domain take into account wishes with respect to starting times for lectures and seminars (constraint C 12). In the first step, it is attempted to reduce the domains of the variables such that the expressed wishes are included in the reduced domain. Instead of preferred time points, we consider intervals that include these points. More specifically, if tp is such a time point, then the interval $[\,tp-2, tp+2\,]$ is generally considered (i.e., a difference of two quarter-hours in each direction). This is the default interval, if no other interval is specified. Another difference may be given when specifying the problem. The time intervals created by preferred time points are called preferred intervals below. The lectures are scheduled before the seminars are. The search methods used for scheduling lectures and seminars are slightly different.

The seminars on a particular subject are held for a given number of groups. Let us suppose that the number of different preferred time points given for the seminars on a particular subject is large enough to allow all these seminars to be easily scheduled at these time points if no other seminars are considered (but the scheduled lectures have to be taken into account). We restrict the starting-time variables of the seminars to the union of the corresponding preferred intervals. Other time points are not considered for these variables. However, we do not restrict the domains in this way if an interactive scheduling of individual courses is carried out (see next section). Based on the given ordering, the domains of all starting-time variables are reduced step by step to a preferred interval in the first search step. Since the domains are reduced to an interval, the different durations and the different breaks needed between two courses can be better taken into account. In particular, the number of backtracking steps can be reduced by this method in comparison with a conventional labelling algorithm. In the second search step, values are selected for all domain variables. For this value selection, a labelling algorithm is used which attempts to select the preferred values first.

The domains of the starting-time variables of the lectures are not reduced to the union of the preferred intervals. In the first search step, we try to reduce

all starting-time variables of lectures to a preferred interval. If such a preferred interval cannot be chosen, then the corresponding variable is reduced to another time interval. Then, in the next search step, values are selected for all domain variables.

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. 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. With regard to the problems 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, and the priorities of the subjects.

5 Interactive Search

The generation of a timetable can also be controlled interactively by the user. The following 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
- 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 may, however, only alter a timetable in such a way that no hard constraints are violated. If an individual course is scheduled or moved, then the values that can be selected without causing any conflict are displayed graphically. These values are also determined using a kind of *forward checking* (see [13] for inference rule *forward checking*). Thus, selection of such a value does not directly result in a contradiction. However, a solution obtained with this selection may not exist because the constraint solver is not complete. The following program describes the basic algorithm used in this check:

```
dom_check(DomVar) :-
   dvar(DomVar),
   !,
   domain(DomVar,ValueList),
   dom_check1(LValue,DomVar).
dom_check(_).

dom_check1([],_).
dom_check1([N|List],DomVar) :-
   not N = DomVar,
   !,
   DomVar #\= N,
   dom_check1(List,DomVar).
dom_check1([N|List],DomVar) :-
   dom_check1(List,DomVar).
```

For a given domain variable `DomVar`, the procedure `domain(DomVar,ValueList)` generates the list of elements `ValueList` belonging to the domain of `DomVar`. The relation `DomVar #\= N` means that `DomVar` and `N` are different and, consequently, that the value `N` is deleted from the domain of `DomVar`.

This interactive component of our timetabling system is very important. Without it, our timetabling system could not be used successfully for the generation medical faculty's timetables. The interactive component 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 generated or partially generated timetable. The timetabling system ensures that no hard constraints are violated by such modifications.

6 Implementation

The Constraint Logic Programming language CHIP was selected as the implementation language. The global constraints and the object-oriented component built into CHIP are particularly useful for problem modelling. The main components of our timetabling system are

graphical editor: specification of a timetabling problem
transformation: transformation of a problem description into an internal representation and transformation of a state into an external description
graphical interfaces: graphical representation of timetables and interactions of the user with the system
search: generation of constraints and solution search
HTML code: generation of HTML code for results

For each main component, there are different parts for the different subproblems relating to lectures, seminars and practical courses, respectively. For instance, there are graphical interfaces for scheduling the lectures, seminars and practical courses. There are common basic dates. The timetables can be generated relatively independently of each other. However, the timetables for the lectures have to be generated first and the scheduled lectures are taken into account when scheduling the seminars.

For representation of a timetabling problem, we used two phases: definition of the problem, and the internal object-oriented representation. For the first phase, definition of the problem, we developed a declarative language for problem description. All the components of a timetabling problem can be easily defined using this declarative language. In the second phase, the internal object-oriented representation is generated from the problem definition. The object-oriented representation is used for the solution search and the graphical user interface. Output of the generated timetables is in HTML format. This means that the results of the timetabling are available for further use elsewhere.

A timetabling problem can be specified using the graphical problem editor or by using the problem description language directly. The problem description language is also used for saving states of scheduling. In this way, incomplete timetables can be saved. Changes in the parameters in the graphical problem editor are also mapped into the problem description language.

The component *search* includes the generation of constraints and all strategies, methods and options of solution search. There are also different schedules for lectures, seminars and practical courses. This made it easier to try out different methods of solution search.

All the components of our timetabling system are implemented in CHIP. About 1.6 Mbytes of program code are used for all these components. Note that the component "*search*" consists of about 190 Kbytes of program code.

7 Results

7.1 Applications

Our timetabling system was well received by the Charité Medical Faculty at the Humboldt University. The timetables were generated quickly and were available at a very early stage. The generation of timetables for this faculty had previously caused problems and involved a great deal of manual work. Since 1998, the faculty's timetables have been generated using our timetabling system. This system has been constantly improved since then.

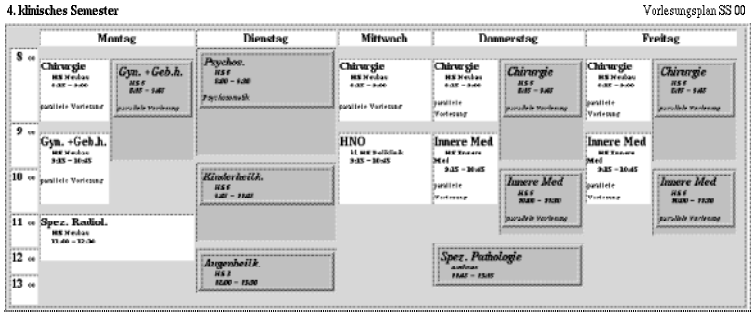

Fig. 1. A timetable from the Internet

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, a solution was found by an interactive step-by-step search within a few minutes. It proved possible to generate a solution for all problems of our application. For different reasons, an initial solution was generally modified interactively . Several variants of a

timetable were able to be tried out within a short time period. The results were very well received by all the departments involved. The importance of a combination of interactive and automatic search was also shown. The generated timetables were output as HTML files and are available in the Internet under `http://www.first.gmd.de/plan/charite`. Figure 1 shows a timetable of lectures from the Internet.

We have successfully used our timetabling system for other applications, too. The timetables of a university and a college's computer science departments were generated within a few seconds. We also generated timetables for a school. Some modifications are necessary, however if our system is used in a school.

7.2 Experimental results

This section presents some experimental results. 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 2 and 3 show two solutions to these problems. Note that these solutions are interactively modified versions of solutions that were generated automatically.

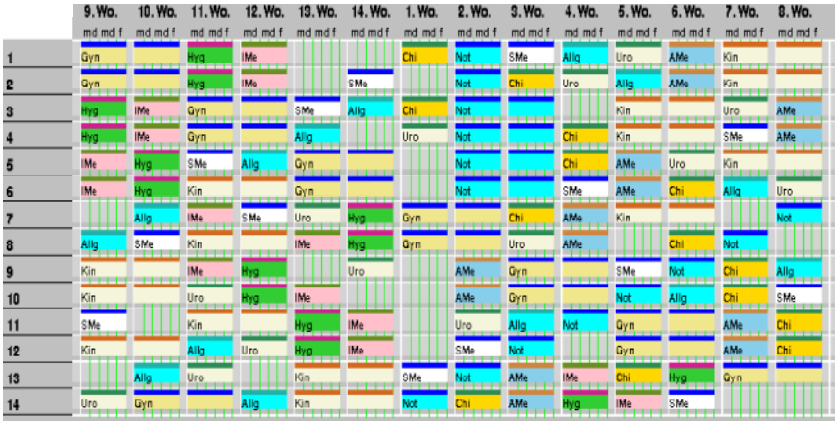

Fig. 2. A solution for the examples *Ex 11* , *Ex 12* , *Ex 13*

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 only difference between the examples *Ex 12* and *Ex 13* is the priority of **one** subject.

For a given ordering of practical courses, we define two different strategies for assigning values to the variables *Start* and *StartNr* for each course:

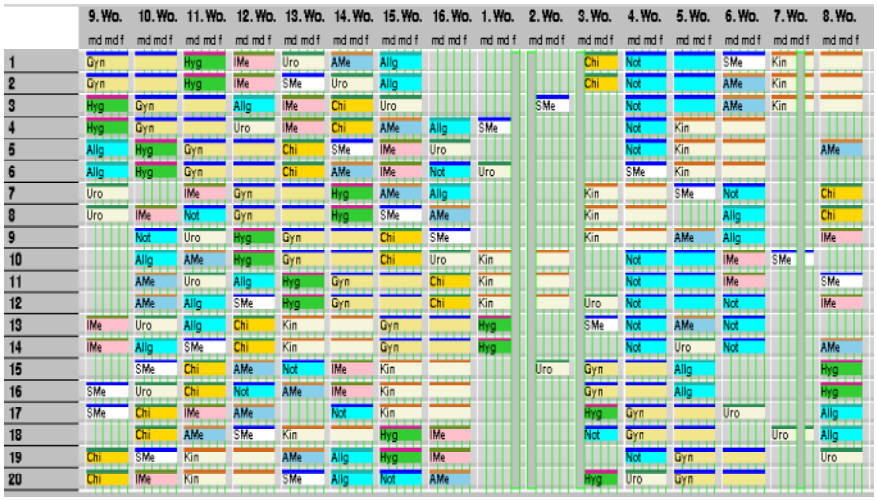

Fig. 3. A solution for the examples *Ex 21* , *Ex 22*

S 1: First, for all practical courses, values are assigned step by step to the variable *StartNr*. Then, values are assigned step by step to the variable *Start*.

S 2: First, for all practical courses, values are assigned step by step to the variable *Start*. Then, values are assigned step by step to the variable *StartNr*.

The following possibilities for adding redundant constraints are considered in this paper (see also Section 3.4):

diffn: for the constraints C 3 and C 4 , additional `diffn` constraints for both kinds of domain variables

among-start: additional `among` constraints for the domain variable *Start*

among-nr: additional `among` constraints for the domain variable *StartNr*

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

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

Table 2 presents some computation results for several methods and the five examples. We use the two strategies S 1 and S 2 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 a small number of 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

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

Table 1. Definition of methods

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.

		Ex 11		*Ex 12*		*Ex 13*		*Ex 21*		*Ex 22*	
method		#	sec	#	sec	#	sec	#	sec	#	sec
	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	259	3.3	–	8.0	246	2.2	0	0.7	0	0.6
S 1	d	0	0.7	–	28.9	14	0.8	0	1.4	0	1.4
	e	259	4.4	–	9.6	242	3.0	0	0.8	0	0.8
	f	0	0.8	–	34.0	14	0.9	0	1.6	0	1.7
	g	–	29.9	–	15.6	–	13.2	33	2.6	69	2.8
	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	25	0.4	–	7.4	84	0.9	–	9.6	–	10.1
S 2	d	25	0.8	34	1.1	36	1.4	–	20.7	–	20.7
	e	1	0.3	–	13.6	84	1.3	0	0.7	–	25.8
	f	1	0.7	10	0.9	23	1.4	0	1.4	–	52.2
	g	1	0.6	11	0.8	44	2.0	0	1.4	–	54.1

Table 2. Comparison of several methods

Interesting conclusions from the experimental results including the results not represented here are:

- There is no one best search strategy. If only one of the above two strategies is selected, then no solution is found for *Ex 12* or *Ex 22*
- The best method for adding redundant constraints is method "f".
- 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*.

– 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 needed 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.

8 Conclusions

Initial application of our timetabling system has proved successful and has demonstrated the suitability of the methods used. From this application, we were able to obtain useful information for our future work. Important conclusions from our experience with automatic timetabling are:

– Constraint Logic Programming is well suited for combined automatic and interactive timetabling.
– The interactive component is very important for successful practical application.
– The chosen method of problem modelling has a considerable influence on the solution search. If a better method of problem modelling is chosen, then the solution search is robuster with respect to the heuristics used for the search.
– Global constraints are very useful for modelling complex conditions.
– The search space can be reduced considerably by adding redundant constraints.
– For solution search, the number of backtracking steps should be restricted and different heuristics should be tried out.
– The search can be improved by using the domain-reducing strategy.
  In particular, wishes regarding starting times and the different length of breaks needed between courses (as a result of their being held at different locations) can be better integrated into the search.
– A declarative problem description language is well suited for problem definition and for saving scheduling states.

Our future research on timetabling problems will include investigations of heuristics for variable selection and continued study of the influence of different modelling techniques on the solution search. The methods, techniques and concepts developed or under development will also be tested on other applications. Our timetabling system is currently being modified for scheduling a company's further education courses.

References

1. S. Abdennadher and M. Marte. University timetabling using constraint handling rules. In O. Ridoux, editor, *Proc. JFPLC'98, Journées Francophones de Programmation Logique et Programmation par Contraintes*, pages 39 –49, Paris, 1998. Hermes.
2. F. Azevedo and P. Barahona. Timetabling in constraint logic programming. In *Proc. World Congress on Expert Systems*, 1994.
3. E. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *J. Mathematical and Computer Modelling*, 20(12):97–123, 1994.
4. P. Boizumault, Y. Delon, and L. Peridy. Constraint logic programming for examination timetabling. *J. Logic Programming*, 26(2):217–233, 1996.
5. E. Burke and M. Carter, editors. *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1998.
6. E. Burke and P. Ross, editors. *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1996.
7. T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In *[6]*, pages 183–295, 1996.
8. T. Frühwirth. Theory and practice of constraint handling rules. *J. Logic Programming*, 37:95–138, 1998.
9. H.-J. Goltz. Reducing domains for search in CLP(FD) and its application to job-shop scheduling. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming – CP'95*, volume 976 of *Lecture Notes in Computer Science*, pages 549–562, Berlin, Heidelberg, 1995. Springer-Verlag.
10. H.-J. Goltz. On methods of constraint-based timetabling. In C. Gervat, editor, *Proceedings PACLP 2000*, pages 167–177. The Practical Application Company Ltd, 2000.
11. H.-J. Goltz and D. Matzke. University timetabling using constraint logic programming. In G. Gupta, editor, *Practical Aspects of Declarative Languages*, volume 1551 of *Lecture Notes in Computer Science*, pages 320–334, Berlin, Heidelberg, New York, 1999. Springer-Verlag.
12. C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In *[6]*, pages 130–145, 1996.
13. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge (Mass.), London, 1989.
14. M. Henz and J. Würtz. Using Oz for college timetabling. In *[6]*, pages 162–177, 1996.
15. M. Kambi and D. Gilbert. Timetabling in constraint logic programming. In *Proc. 9th Symp. on Industrial Applications of PROLOG (INAP'96)*, Tokyo, Japan, 1996.
16. G. Lajos. Complete university modular timetabling using constraint logic programming. In *[6]*, pages 146–161, 1996.
17. K. Marriott and P. J. Stucky. *Programming with Constraints: An Introduction.* The MIT Press, Cambridge (MA), London, 1998.
18. A. Schaerf. A survey of automated timetabling. Technical Report CS-R9567, Centrum voor Wiskunde en Informatica, 1995.
19. G.M. White and J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. In *[5]*, pages 187–198, 1998.