University Timetabling Using Constraint Logic Programming *

Hans-Joachim Goltz and Dirk Matzke

GMD – German National Research Center for Information Technology
GMD-FIRST, Rudower Chaussee 5, D-12489 Berlin
goltz@first.gmd.de

Abstract. A timetable is a temporal arrangement of a set of meetings such that all given constraints are satisfied. A timetabling problem can be suitably modelled in terms of a set of constraints. We use Constraint Logic Programming and develop methods, techniques and concepts for a combination of interactive and automatic timetabling of university courses and school curricula. An exemplary application of such a timetabling system was developed for the Charité Medical Faculty at the Humboldt University, Berlin. The timetabling system is flexible enough to take into account special user requirements and to allow constraints to be modified easily if no basic conceptual change in the timetabling is necessary. 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.

1 Introduction

Constraint Logic Programming over finite domains is a rapidly growing research field aiming at the solution of large combinatorial problems. For many real-life problems, the Constraint Logic Programming approach has been applied with great success. 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. Besides the so-called hard constraints, all of which must be satisfied, there are also constraints which should be satisfied as far as possible; these are termed soft constraints. A timetabling problem can be suitably modelled in terms of a set of constraints. Constraint Logic Programming allows formulation of all constraints in a declarative manner.

Timetabling has long been known to belong to the class of problems called NP-complete (see e.g. [9]). Existing software products for timetabling impose too severe restrictions on the kind of timetables that can be generated. In addition, they are not flexible enough to allow anomalous requirements to be integrated or the special features found in virtually every user organization to be taken into account (see e.g. [7]).

* in: G. Gupta (ed.), Practical Aspects of Declarative Languages, LNCS 1551, Springer-Verlag 1999, pp. 320-334

Automated timetabling is a current and relevant field of research. The Second International Conference on Automated Timetabling was held in 1997 ([8, 6]). Different software methods and approaches are available for automated timetabling (see e.g. [19]). Various Constraint Logic Programming approaches are discussed in [1, 3, 5, 13, 15, 17, 16, 20].

In [3], 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 (w.r.t. other such languages). The timetabling problems of a computer science department are also addressed in [16] and [1], the former using the Constraint Logic Programming language ECL^{i}PS^e and the latter selecting the high-level constraint language Constraint Handling Rules (CHR) as the implementation language. The language CHR was developed by Tom Frühwirth [11]. Based on an existing domain solver written in CHR, a solver for the timetabling problem was developed, requiring no more than 20 lines of code. [5] and [13] look at the application of the Constraint Logic Programming language CHIP to a university's examination timetabling and describe the use of the global constraint *cumulative*. Different labelling strategies are discussed in [13].

M. Henz and J. Würtz [15] present a solution to 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 for functional, object-oriented and constraint programming. G. Lajos [17] 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 [20], 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 combination of interactive and automatic timetabling of university courses and school curricula. The automated solution search will be implemented in such a way that, it normally allows a solution to be found in a relatively short time, if such a solution exists. The timetabling systems will be flexible enough to take into account special user requirements and to allow constraints to be modified easily, if no basic conceptual change in the timetabling is necessary. 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.

The research focuses on the following areas: concepts and methods for modelling timetabling problems; study 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 goals are to be met using Constraint Logic Programming, this being the approach best suited to our

needs. The methods, techniques and concepts developed or under development are tested and further elaborated on prototypical applications.

An exemplary application of a combined interactive and automatic course-timetabling system was developed for the Charité Medical Faculty at the Humboldt University, Berlin. This system is called *CharPlan* and realizes timetabling by means of Constraint Logic Programming and a special algorithm for an efficient search. The first test phase has been successfully completed. The generated timetables were output as HTML files and are available on the Internet.

In this paper, we describe the generation of this timetabling system. Section 2 gives a brief description of the timetabling problem. This is following by a discussion of Constraint Logic Programming, problem representation and search methods. Some remarks on the implementation and results are also given.

2 Problem Description

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, therebeing between 12 and 24 groups per semester. The buildings of the Charité Medical Faculty at the Humboldt University are at two different locations in Berlin. Some practicals are also carried out in clinics and hospitals in other parts of Berlin. Other important constraints for this timetabling problem are:

C_1 : The courses can begin at every quarter hour and may run for different lengths of time.
C_2 : There are restrictions with respect to the starting time for each course.
C_3 : Two lectures on a same subject may not be scheduled for the same day.
C_4 : Each group's lectures, seminars and practicals should not overlap.
C_5 : The number of seminars and practicals a department (research group) held at any one time is limited.
C_6 : Some courses are only held during certain weeks, the following arrangements are possible: every week, A-weeks (weeks with odd numbers: 1,3,5, ...), B-weeks (weeks with even numbers: 2,4,6, ...), either A-weeks or B-weeks.
C_7 : Lectures on the same subject are carried out in parallel at different parts of the faculty. Thus a student can choose which of the parallel lectures to attend.
C_8 : Timetabling also involves allocating rooms for the individual lectures.
C_9 : Some lectures can only be held in certain specified rooms; in some cases only one room matches the requirements.
C_{10} : The varying distances between the different course locations must be taken into account.
C_{11} : The starting time preferences for courses and the preferred rooms for lectures should also be taken into account where possible.

The constraint C_{11} is a soft constraint. The other constraints are hard constraints and must be satisfied.

3 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. In the Constraint Logic Programming paradigma, a constraint satisfaction problem can be represented by Horn clause logic programs in which the clause bodies may contain constraints. Constraints are generated incrementally during runtime and passed to a constraint solver which applies domain-dependent constraint satisfaction techniques to find a feasible solution for the constraints. Constraint Logic Programming with constraints over finite integer domains, CLP(FD), has been established as a practical tool for solving discrete combinatorial problems.

The Constraint Logic Programming language CHIP ([10]) was one of the first CLP languages, together with PROLOG III and CLP($\mathcal{R}$). It was developed at the European Computer-Industry Research Centre (ECRC) in Munich between 1985 and 1990, and is now being developed and marketed by the French firm COSYTEC. The current version of CHIP differs in particular from the earlier constraint systems by the inclusion of global constraints ([2, 4]). The CHIP system contains different constraint solvers. We only use constraints over finite domains. Some built-in constraints of the CLP(FD) part of CHIP are briefly described below.

Each constrained variable has a domain that must first be defined. Such a variable is also called domain variable. The usual arithmetic constraints *equality*, *disequality*, and *inequalities* can be applied over linear terms built upon domain variables and natural numbers. The *alldifferent* constraint is a symbolic constraint and expresses a constraint that all members of a list of domain variables and integers must have different values. Instead of this constraint, a logical equivalent conjunction of disequalities could be used. The disadvantage of using a conjunction of disequalities is that consistency methods for such a conjunction are quite weak as the disequalities are considered in isolation. Symbolic constraints can be used for more complex conditions concerning the relation between several domain variables. A change in the domain of one variable may have an effect on all other variables occurring in the constraint. A very useful symbolic constraint is `element(N,List,Value)`. It specifies that the $\texttt{N}^{th}$ 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.

Global constraints use domain-specific knowledge to obtain better propagations results and can be applied to large problem instances. 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. The basic ideas of the global constraints *diffn* and *cumulative* are given in the rest of this section.

The *cumulative constraint* was orginally introduced to solve scheduling and placement problems ([2]). The simplified form of this constraint is

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

where $[S_1, S_2, \ldots, S_n]$, $[D_1, D_2, \ldots, D_n]$, and $[R_1, R_2, \ldots, R_n]$ are nonempty lists of domain variables (or natural numbers) and L is a natural number. The usual interpretation of this constraint is a single resource scheduling problem:

> *Each S_i represents the starting time of some event, D_i is the duration and R_i the amount of resources needed by this event. L is the total amount of resources available at each relevant point of time. The cumulative constraint ensures that, at each time point in the schedule, the amount of resources consumed does not exceed the given limit L.*

The mathematical interpretation of this constraint is:

> for each $k \in [min\{S_i\}\,,\ max\{S_i + D_i\} - 1]$ it holds that:
> $\Sigma\, R_j \ \leq\ L$ for all j with $S_j \leq k \leq S_j + D_j - 1$

The *diffn constraint* was included in CHIP to handle multidimensional placement problems. The basic diffn constraint takes as arguments a list of n-dimensional rectangles, where origin and length can be domain variables with respect to each dimension. An n-dimensional rectangle is represented by a tuple $[X_1, \ldots, X_n, L_1, \ldots, L_n]$, where X_i is the origin and L_i the length of the rectangle in the i^{th} dimension. This constraint ensures that the given n-dimensional rectangles do not overlap. Figure 1 gives an example of three two-dimensional rectangles with the nonoverlapping constraint. The largest rectangle, for instance, is determined by the second list `[1,2,3,2]`: the origin is the point `(1,2)`, and the lengths of the two dimensions are `3` and `2`.

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.

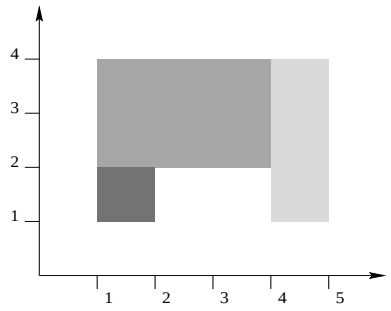

Fig. 1. `diffn([[1,1,1,1],[1,2,3,2],[4,1,1,3]])`

4 Problem Representation

A timetabling problem can be suitably modelled in terms of a set of constraints. Since constraints are used, the problem representation is declarative. In this section, the problem representation is described briefly. The method chosen for problem representation has a considerable influence on the solution search. The success of the search often depends directly on the model chosen. Furthermore, 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.

For each course the *starting time* is represented by a domain variable. If we assume that 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.

With the exception of the constraint C_{11}, all other constraints of the timetabling problem are directly represented by built-in constraints. C_{11} is a soft constraint and is integrated into the solution search. Conditional constraints are not used.

The constraints C_1 and C_2 can be easily satisfied by the choice of the domains for the variables of *starting time* and by corresponding restrictions of these domains. Analogously, the constraint C_9 can be satisfied. Note that the capacity of a room is taken into account by the choice of the domain of a room variable.

If two lectures are not scheduled for the same day, 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 `S1 in 1..240` and `S2 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[1] `S1 #< X` and `X #< S2` ensure the constraint C_3. This method of modelling is more efficient than the method suggested in [13] for the same constraint. In [13], for each lecture, the domain variables `Day in 1..5` and `Hour in 1..48` are considered additionally (`X` is not considered) and the constraint `S #= 48*(Day-1) + Hour` is generated. Then, the constraint C_3 is modelled using the predicate *alldifferent* with respect to the `Day`-variables. The following example shows the different propagation properties. If we consider the question

[1] The arithmetic relations of the finite domain solver are indicated by `#` in CHIP

```
?-  Day1 in 1..5,  Hour1 in 1..48,  S1 in 1..240,
    S1 + 48 #= 48*Day1 + Hour1,
    Day2 in 1..5,  Hour2 in 1..48,  S2 in 1..240,
    S2 + 48 #= 48*Day2 + Hour2,
    alldifferent([Day1,Day2]),
    Day1 = 3.
```

we get the answer:

```
 Day1 = 3
Hour1 = Hour1 in {1..48}
   S1 = S1 in {97..144}
 Day2 = Day2 in {1..2,4..5}
Hour2 = Hour2 in {1..48}
   S2 = S2 in {1..240}
```

The corresponding question for our method is:

```
?-  S1 in 1..240,  S2 in 1..240,
    X in [48,96,144,192],
    S1 #<= X,  X #< S2,
    96 #< S1,  S1 #<= 144.
```

In this case, `Day1 = 3` is modelled by the two constraints `96 #< S1` and `S1 #<= 144`. We get the following answer to this question:

```
   S1 = S1 in {97..144}
   S2 = S2 in {145..240}
    X = X in {144,192}
```

With our method, the domain of the variable `S2` is reduced more than with the method suggested in [13]. However, our method can only be used if the two lectures have the same features. The method given in [13] can also be used in other cases. We can extend our method to cover the case that the durations (`d1`, `d2`) of the lectures are different. In this case, domain variables are defined for duration: `D1 in [d1,d2]` and `D2 in [d1,d2]`. These variables are linked by the constraint `D1 + D2 #= d1 + d2`.

The constraints C_4 and C_5 can be represented by cumulative constraints. For example, let `S1,S2,...,Sn` be the domain variables of starting times of the courses held by a certain department and let `D1,D2,...,Dn` be the corresponding lengths of time. If `Max` is the maximum number of courses that this department can give at any one time, then the constraint C_5 can be modelled by[2]:

```
cumulative([S1,S2,...,Sn], [D1,D2,...,Dn], [1,1,...,1], Max)
```

For each group, the constraint C_4 can be modelled analogously, where `Max` being equal to 1 in this case.

[2] The unused arguments are not mentioned.

The use of diffn constraints can ensure that the constraints C_6 and C_7 are satisfied, a course being represented by a "3-dimensional rectangle" with the dimensions *starting time*, *week* and *parallelism*. If the allocation of rooms is required, then the attribute *room* has to be considered as a fourth dimension (C_8).

In the following examples, we assume that *week* and *parallelism* do not need to be considered. Let `S1,S2,...,Sn` be the domain variables of starting times, `D1,D2,...,Dn` be the corresponding durations, and let `R1,R2,...,Rn` be the domain variables for room allocations. The constraint C_8 can be modelled by the following constraint, where the length of the dimension *"room"* is 1 unit:

```
diffn([[S1,D1,R1,1],[S2,D2,R2,1],...,[Sn,Dn,Rn,1]])
```

This constraint can also be modelled by the cumulative constraint, where additional domain variables `X1,X2,...,Xn` 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 [5].

We consider a simple example of these two methods: 4 lectures (the first four) with a duration of 2 time units, 2 lectures with a duration of 1 time unit, and two rooms (1 and 2). The lectures are scheduled within 6 time units, and the second and the fifth lectures cannot be allocated to room 1. Furthermore, we assume that the first lecture is scheduled at time point 3 in room 2. This example can be modelled with the diffn constraint as follows:

```
?- [S1,S2,S3,S4] in 1..5,  [S5,S6] in 1..6,
   [R1,R2,R3,R4,R5,R6] in 1..2,  R2 #\= 1,  R5 #\= 1,
   diffn([[S1,R1,2,1],[S2,R2,2,1],[S3,R3,2,1],
          [S4,R4,2,1],[S5,R5,1,1],[S6,R6,1,1]]),
   R1=2,S1=3.
```

The answer to this question is:

```
S1 = 3,                          R1 = 2,
S2 = S2 in {1,5},                R2 = 2,
S3 = S3 in {1..5},               R3 = 1,
S4 = S4 in {1..5},               R4 = 1,
S5 = S5 in {1..2,5..6},          R5 = 2,
S6 = S6 in {1..6},               R6 = R6 in {1..2}
```

If the cumulative constraint[3] is used, then this question can be modelled by:

```
?- [S1,S2,S3,S4] in 1..5,  [S5,S6] in 1..6,
   [R1,R2,R3,R4,R5,R6] in 1..2,  R2 #\= 1,  R5 #\= 1,
   [X1,X2,X3,X4,X5,X6] in 11..26,
   X1 #= 10*R1 + S1,  X2 #= 10*R2 + S2,
   X3 #= 10*R3 + S3,  X4 #= 10*R4 + S4,
   X5 #= 10*R5 + S5,  X6 #= 10*R6 + S6,
   cumulative([X1,X2,X3,X4,X5,X6],[2,2,2,2,1,1],[1,1,1,1,1,1],1),
   X1 = 23.
```

[3] The unused arguments are not mentioned.

In this case, the answer to this question is:

```
S1 = 3,                          R1 = 2,
S2 = S2 in {1..5},               R2 = 2,
S3 = S3 in {1..5},               R3 = R3 in {1..2},
S4 = S4 in {1..5},               R4 = R4 in {1..2},
S5 = S5 in {1..6},               R5 = 2,
S6 = S6 in {1..6},               R6 = R6 in {1..2},

X1 = 23,                       X2 = X2 in {21,25},
X3 = X3 in {11..21,25},        X4 = X4 in {11..21,25},
X5 = X5 in (21..22,25..26},    X6 = X6 in {11..22,25..26}
```

The domains of the variables are more reduced if the conditions are defined using the diffn constraint. In particular, the constraint solver then determines that `R3` and `R4` can only be equal to 1.

The diffn constraints can also be used for constraint C_{10}, where at least the dimensions *starting time* and *location* are used. 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* and depends on the selected location of the corresponding course.

We consider a simple example of this kind of modelling. Let A and B be two lectures that can be held at the two locations 1 and 2. To allow for the distance between different locations, a break of 3 time units must be scheduled, and for the same location, a break of 1 time unit is sufficient. The duration of these lectures is assumed to be 4 time units. Let `Sa`, `Sb` be the variables for the starting time, and `La`, `Lb` the variables for the location of these lectures. In this example, we assume that lecture B is scheduled after lecture A, and we consider only the dummy courses $A1$ and $A2$ for the breaks needed after lecture A. The variables for the starting times of these dummy courses are denoted by `S1` and `S2`, respectively. `D1` and `D2` denote the variables for the duration of these activities. Then, the constraints can be implemented by:

```
example(Sa,La,Sb,Lb) :-
    [Sa,Sb] in 1..20,
    [S1,S2] in 1..20,
    [La,Lb] in 1..2,
    element(La,[1,3],D1),
    element(La,[3,1],D2),
    Sa + 4 #= S1,  Sa + 4 #= S2,
    diffn([[Sa,La,4,1],[S1,1,D1,1],[S2,2,D2,1],[Sb,Lb,4,1]]),
    Sa + 4 #<= Sb.
```

We assume that lecture A is scheduled at time point 1. The following questions demonstrate the interdependence of the variable `Sb` and the choice of the locations `La, Lb`.

```
?- example(Sa,La,Sb,Lb), Sa=1,La=1,Lb=1.    ==>  Sb = Sb in {6..20}
?- example(Sa,La,Sb,Lb), Sa=1,La=1,Lb=2.    ==>  Sb = Sb in {8..20}
?- example(Sa,La,Sb,Lb), Sa=1,La=1,Sb=6.    ==>  Lb = 1
?- example(Sa,La,Sb,Lb), Sa=1,La=2,Lb=1.    ==>  Sb = Sb in {8..20}
?- example(Sa,La,Sb,Lb), Sa=1,La=2,Lb=2.    ==>  Sb = Sb in {6..20}
?- example(Sa,La,Sb,Lb), Sa=1,La=2,Sb=7.    ==>  Lb = 2
?- example(Sa,La,Sb,Lb), Sa=1,Sb=7,Lb=2.    ==>  La = 2
```

Figure 2 shows the schedule of lecture A and the dummy courses for the two different location choices (`La` and `Lb`, respectively).

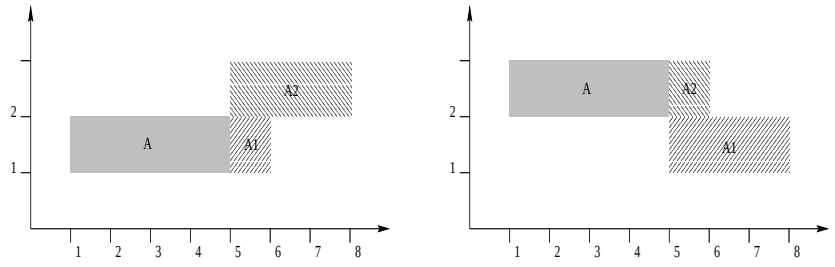

Fig. 2. Lecture A with dummy courses for the breaks

5 Search Methods

A solution of a timetabling problem is a scheduling of the given courses such that all hard constraints are satisfied. The soft constraints should be satisfied as far as possible. 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:

- select a variable from the set of problem variables considered, choose a value from the domain of this variable and assign this value to the variable
- consistency techniques are used by the constraint solver in order to compute the consequences of this assignment; 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

We propose a generalization of the labelling method presented in [12]. 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. The following program describes the basic algorithm of this search:

```
reducing( [], _ ).
reducing(VarList, InfoList) :-
    select_var(Var,VarList,NewVarList),
    reducing_domain(Var, InfoList),
    reducing(NewVarList, InfoList).
```

Domain reduction of a selected variable may be dependent on different components. Thus, we assume that the second argument of `reducing_domain/2` contains the information required for determining the reduced domain. A reduced domain should be neither too small nor too large. A solution is narrowed 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 the final value assignment, the search can backtrack into the reducing procedure.

The domain-reducing strategy is also used in our timetabling system. The advantages of this method have been shown by many examples.

In the domain-reducing strategy, the variable selection and the choice of a reduced domain for the selected variable are nondeterministic. The success of the domain-reducing strategy depends on the chosen heuristics for the order of variable selection and for the determination of the reduced domain. The order of variable selection is determined by the following characteristics of the courses: the given priority of the related subject, and whether the assignment of the starting time or the room has priority. If priority of a subject is not given, then the priority is generated by a random number. The complete order of variable selection is determined by random numbers if necessary. The chosen heuristics for determining the reduced domain take into account wishes with respect to starting times (see constraint C_{11}). Thus, 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.

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 orderings of variable selections are tried out.

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
- scheduling the remaining courses automatically

These actions can be perfomed in any order or combination, and no automatic backtracking is caused such user actions. The user can, however, only alter a timetable in such as 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 graphically represented. These values are also determined using the inference rule *forward checking* (see [14] for this rule). Thus, selection of such a value does not directly result in a contradiction. However, a solution with this selection must not exist because the constraint solver is not complete.

Figure 3 shows a detail of the graphical user interface during scheduling of an individual course. At the bottom of the picture, the time points that can be selected and the preferred starting times (in this case one only) are marked.

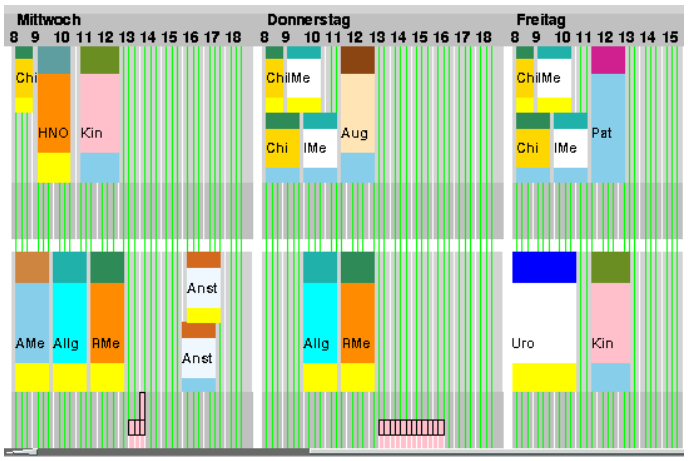

Fig. 3. Detail of the graphical user interface

6 Implementation and Results

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.

For representation of the problem, we used three phases: definition of the problem, the internal relational representation, and the internal object oriented representation. For the first phase, definition of the problem, we developed a declarative language for problem description. All components of a timetabling problem can be easily defined using this declarative language. Thus, the graphical user interface is only needed to set the parameters. In the second phase, the problem definition is transformed into an internal relational representation. The definition data are converted into a structure that is suitable for finite integer domains. In the third phase, the internal object-oriented representation is generated from the internal relational representation. The object-oriented representation is used for the solution search and the graphical user interface.

The transformation process takes into accout the possible cross connections between the definitions and supports the generation of various solutions. The results of the second phase can be used to generate the corresponding objects needed for the different solutions.

Output of the generated timetables is in HTML format. This means that the results of the timetabling are available for further use elsewhere.

In the first phase, the timetables are generated for the clinical-course phase of the studies program (six semesters). Generation of timetables for this course phase had previously caused problems, and involved a great deal of manual work.

The first test phase has been successfully completed. The timetables were generated quickly if there were not any contradictions (violations of hard constraints). Such contradictions were easily detected using the interactive graphical user interface. Conflicts were eliminated in collaboration with the person responsible for timetables at the medical faculty. Knowledge of this field was very important for removing contradictions to ensure that wishes were satisfied in most cases. The results were very well received by all the departments involved.

The generated timetables were output as HTML files and are available in the Internet under `http://www.first.gmd.de/plan/charite`. Figure 4 shows a timetable of lectures from the Internet.

Our timetabling system was given a positive reception by the Charité Medical Faculty at the Humboldt University, Berlin. The timetables were generated quickly and were available students at a very early stage. Allocation of students to course groups was done during the preceding semester, the wishes of the students being taken into account whenever possible.

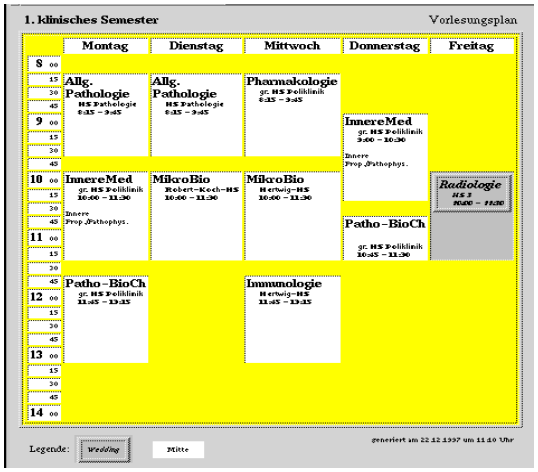

Fig. 4. A timetable from the Internet

7 Conclusions and Further Work

The initial application of our timetabling system was proved successful and demonstrating the suitability of the methods used. From this application, we were able to obtain useful information for our further work. The importance of a combination of interactive and automatic search was also shown. Further development of our timetabling system will include scheduling courses for any semester, allocation of rooms for all courses, improvements in problem modelling and solution search, and perfection of the graphical user interface. Research is needed to complete our interactive, automated timetabling system for the medical faculty. Studying of the influence of different modelling techniques on the solution search and exploring techniques for heuristic solution search will be the main elements of this research. The methods, techniques and concepts developed or under development will also be tested on other applications.

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 Constraintes*, pages 39 –49, Paris, 1998. Hermes.
2. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *J. Mathematical and Computer Modelling*, 17(7):57–73, 1993.
3. F. Azevedo and P. Barahona. Timetabling in constraint logic programming. In *Proc. World Congress on Expert Systems*, 1994.

4. E. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *J. Mathematical and Computer Modelling*, 20(12):97–123, 1994.
5. P. Boizumault, Y. Delon, and L. Peridy. Constraint logic programming for examination timetabling. *J. Logic Programming*, 26(2):217–233, 1996.
6. 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.
7. E. Burke, D. Eiiliman, P. Ford, and R. Weare. Examination timetabling in British universities: A survey. In *[8]*, pages 76–90, 1996.
8. 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.
9. T. B. Cooper and J. H. Kingston. The complexity of timetable construction problems. In *[8]*, pages 183–295, 1996.
10. M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Int. Conf. Fifth Generation Computer Systems (FGCS'88)*, pages 693–702, Tokyo, 1988.
11. T. Frühwirth. Theory and practice of constraint handling rules. *J. Logic Programming*, 37:95–138, 1998.
12. 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.
13. C. Guéret, N. Jussien, P. Boizumault, and C. Prins. Building university timetables using constraint logic programming. In *[8]*, pages 130–145, 1996.
14. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge (Mass.), London, 1989.
15. M. Henz and J. Würtz. Using Oz for college timetabling. In *[8]*, pages 162–177, 1996.
16. M. Kambi and D. Gilbert. Timetabling in constraint logic programming. In *Proc. 9th Symp. on Industrial Applications of PROLOG (INAP'96)*, Tokyo, Japan, 1996.
17. G. Lajos. Complete university modular timetabling using constraint logic programming. In *[8]*, pages 146–161, 1996.
18. K. Marriott and P. J. Stucky. *Programming with Constraints: An Introduction.* The MIT Press, Cambridge (MA), London, 1998.
19. A. Schaerf. A survey of automated timetabling. Technical Report CS-R9567, Centrum voor Wiskunde en Informatica, 1995.
20. G.M. White and J. Zhang. Generating complete university timetables by combining tabu search with constraint logic. In *[6]*, pages 187–198, 1998.