Building University timetables using Constraint Logic Programming

Christelle Guéret[1,2], Narendra Jussien[1], Patrice Boizumault[1], Christian Prins[1]
email: {Christelle.Gueret, Narendra.Jussien,
Patrice.Boizumault, Christian.Prins}@emn.fr

[1] École des Mines de Nantes
4 Rue Alfred Kastler, La Chantrerie, F-44070 Nantes Cedex 03, France.
[2] Institut de Mathématiques Appliquées
3 Place André Leroy, BP 808, F-49008 Angers Cedex 08, France.

1 Introduction

A timetabling problem can be defined as the scheduling of a certain number of lectures, which are to be attended by a specific group of students and given by a teacher, over a definite period of time. Each lecture requires certain resources (rooms, overheads,...) in limited number and must fulfill certain specific requirements. In particular, automatic building of timetables is extremely difficult because of the diversity of the constraints that must be taken into account.

The most usual methods to solve this problem are inherited from Operations Research such as graph coloring [14, 15, 19, 20] and mathematical programming [19, 20], from local search procedures such as simulated annealing and Tabu search [30, 31, 10, 11] or from Genetic Algorithms [13]. These well-known and widely used methods have given good results. But, OR inherited methods generally lack flexibility (i.e modifying the data may lead to the necessity of reconsidering the initial model); moreover it is difficult to find a model which includes all the constraints. For local search methods (where most of the constraints are put in the objective function) or for Genetic Algorithms (where the constraints are active in the fitness function), the user frequently obtains solutions by tuning rather than by defining his own search strategy dedicated to the problem.

Constraint Logic Programming is based upon the integration of Constraint Solving and Logic Programming. This combination helps make Constraint Logic Programming programs both expressive and flexible, and in some cases, more efficient than other kinds of programs.

Constraint Logic Programming over Finite Domains (CLP) is based upon the integration of CSP (*Constraint Satisfaction Problems*) approach in a Logic Programming scheme (Prolog). It benefits from the results obtained in the AI community in CSP (domains, consistency techniques, filtering algorithms, search strategies, ...) embedded in a Logic Programming scheme. Thus, the user is provided with a uniform framework in order to both model his problem (constraints) and develop his own search methods (labeling). It has already been proved that CLP is successful in tackling many combinatorial optimization prob-

lems such as planning, scheduling, resource allocation, assignment problems, ... [4, 7, 17, 21, 22, 23, 38, 39].

The aim of this paper is to show how CLP) is well suited for solving Timetabling Problems. Particularly, the declarativity of the CSP formalism improves the flexibility (heterogeneous constraints can be directly handled). Moreover, the CLP paradigm gives the ability to rapidly design efficient search procedures.

In this paper, we first present different approaches used for solving timetabling problems. Then, we describe the Timetabling problem of our institute in our University. After a brief description of the CLP programming language CHIP, we present our approach for solving timetabling problems. Finally, we give some hints on how to relax constraints.

2 Solving timetabling problems

General timetabling problems can be defined as the scheduling of a set of lectures which several groups of students must attend, over a preset period of time, using certain resources and satisfying a certain set of constraints. Many researchers [15, 20] have dealt with this problem since the fifties. And nowadays this problem continues to be studied because of its variety and its complexity [12, 46, 47]. In this section, we present OR solution techniques to solve this problem, the genetic algorithms approach, an example of Tabu search and some work in the CSP community.

2.1 OR approaches

Graph Coloring. The timetabling problem which consists in scheduling a set of lectures over p periods is equivalent to a graph vertex coloring problem [20, 15, 14, 19] where:

- Each lecture (a pair $(class, teacher)$) makes a vertex,
- Two vertices are connected if the associated lectures share a student or a teacher.

Then, we must find a vertex coloration using a maximum of p colors. Vertices colored with a same color are associated with classes which take place at the same time. We know that the problem of the existence of a graph vertex p-coloring (using atmost p colors) is $\mathcal{NP}$-Complete when $p \geq 3$.

We cannot introduce easily all types of constraints (e.g. precedence among lectures, allocation of a room to each lecture) in this modeling. Furthermore, crucial information about the nature of the constraints is lost. An edge can represent the fact that a student must attend two lectures, as well as the fact that two lectures must take place in the same room or that they need the same teacher.

Flow problems. Edge graph coloring problems can be solved with heuristics concerning max-flow problems [19, 20]. In this case, we assign to the problem a directed graph the vertices of which are the classes and the teachers. We create an arc (C_i, T_j) if the teacher T_j gives a lecture to the class C_i. We introduce a vertex s and arcs (s, C_i) for each C_i as well as a vertex t and arcs (T_j, t) for each T_j. A path from s to t is associated to a lecture.

Each arc receives a lower and an upper bound (see table 1) which change during the resolution of the problem. We have to solve p max-flow problems in this network (one for each period).

arcs	lower bounds	upper bounds
(s, C_i)	0	1 if all the lectures attended by class C_i are scheduled 0 otherwise
(C_i, T_j)	0	1 if the lecture given by teacher T_j to class C_i is scheduled 0 otherwise
(T_j, t)	0	1 if the lectures given by teacher T_j are scheduled 0 otherwise

Table 1. Lower and upper bounds for the flow modeling

Such a method is interesting because we know efficient algorithms for solving that type of problems. However, when solving a timetabling problem over T periods of time we must find T flows, one for each period. This problem is now again $\mathcal{NP}$-Complete. Furthermore, we cannot introduce all types of constraints (e.g. precedence constraints). Finally, this method demands the same duration for all lectures, which does not seem very realistic.

Mathematical programming. A third method that can be used is mathematical programming with integer variables [19, 20]. The main problem in this case is the number of data and constraints. Optimal resolution of an integer linear program is $\mathcal{NP}$-Hard in the general case, except if the constraint matrix is, for example, totally unimodular (rarely in real life).

For example, consider the timetabling problem which consists of m teachers and n classes. Each teacher T_j gives R_{ij} lectures to the class C_i. The lectures must be scheduled in ε periods of time. Let x_{ijk} be the variable whose value is 1 if the teacher T_j gives a lecture to the class C_i at time k and 0 otherwise. The timetabling constraints are:

$$\sum_{k=1}^{\varepsilon} x_{ijk} = r_{ij}, i =]n] \wedge j =]m] \tag{1}$$

$$\sum_{i=1}^{n} x_{ijk} \leq 1, j =]m] \wedge k =]\varepsilon] \quad (2)$$

$$\sum_{j=1}^{m} x_{ijk} \leq 1, i =]n] \wedge k =]\varepsilon] \quad (3)$$

Equation (1) models that *all the lectures must be scheduled*, and equation (2) that *a teacher cannot give more than one lecture at a time*. Equation (3) ensures that *a student cannot attend more than one lecture at a time*.

In order to reduce the problem size, it is possible to redefine the variables by grouping students, rooms or lectures. [41, 42] propose such a transformation. Next, they use a Lagrangean Relaxation method. However, this method which is quite complex only applies to the specific problem we want to solve, in spite of a certain reduction of the problem complexity. If we add constraints or modify the problem, we must reconsider the entire analysis.

2.2 Genetic Algorithms

We present here the works of [13]. In analogy with the reproduction of living beings, the process starts with an initial population of solutions.

A random population of feasible timetables is created thanks to a graph coloring algorithm [13]. Each timetable is evaluated according to a set of criteria e.g. the length of the timetable, how many students have to sit two exams in a row or how many unused seats there are. Each solution is coded with a chromosome: a vector of symbols whose length is $2N$ (number of exams), divided in N contiguous pieces containing two genes. The two genes of the i^{th} piece represent the period and the room of the exam number i.

The mutation operator randomly changes the period and the room the exam is to be held in, still maintaining a feasible timetable. The cross-over operator takes a pair of timetables, selecting the early exams from one and the late exams from the other to produce a new timetable. A new population is thus generated. The process will be repeated until a good solution is found.

The fitness function includes three criteria: the length of the timetable, the number of conflicts (when a student must take two exams in a row), the spare capacity in each of the rooms.

Genetic algorithms can have good results [13] and find efficient solutions. However, all the parameters must be determined through experimentation. Furthermore, they do not have any guarantee of convergence.

2.3 Tabu search

Tabu search [30, 31] is an effective local search method which moves step by step from one initial solution of a combinatorial optimization problem towards a solution which is expected to be optimal or near-optimal. For each solution s, such a method requires the definition of a neighborhood $V(s)$, consisting of solutions which can be reached in one step from s. The basic step is to move

from the current solution s to the best solution s^* of $V(s)$, even if it is not better than s. A tabu list T is used to avoid cycling. It acts as a short-term memory by storing a description of the NT last moves or solutions. When exploring $V(s)$ to find s^*, T is scanned to avoid the so-called tabu moves which could bring the search back to a previous iteration. The procedure stops after a maximum number of iterations and outputs the best solution found.

J.P. Boufflet [10, 11] implements such a method to solve a timetabling problem. He transforms his problem into a graph coloring problem in which the vertices are the lectures and the edges the constraints. A weight is assigned to each constraint. The problem is then to find a p-coloring which minimizes a multiobjective function. To do this, Boufflet uses a tabu search on a graph coloring problem inspired by de Werra's techniques [32]: each vertex is given one of the p allowed colors. This assignment may not be a coloring (if some adjacent vertices are assigned the same color). The idea is to minimize the conflicts (two adjacent vertices with the same color) by exploring the neighborhood of the initial solution (modification of the color of a node) .

2.4 Constraint Satisfaction Problems

A CSP (Constraint Satisfaction Problem) can be defined as:*a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take. The task is to assign a value to each variable satisfying all the constraints.* [43]

Several methods are used to solve CSP, from filtering algorithms to hybrid algorithms [26, 43]. We will see in the following sections how to use CSP techniques to solve real timetabling problems.

3 Our problem

The IMA (Institute of Applied Mathematics) provides a five years training after the French Baccalaureate. The first three years require a weekly timetable whereas the last two are yearly organized. In order to simplify the presentation, we will only speak about the weekly timetables.

The first three years represent 160 students (60-50-50). Most of the taught subjects are divided into lectures, practical classes given by teachers (called TDE) and practical classes given by students (called TDM). A 4 hour period is reserved each week for examination. A few lectures are independent of our institute. Complete groups attend lectures, whereas half- or one-third groups attend TDE and TDM. Finally, all lectures take place in 8 different rooms with various capacities.

The whole problem can be stated according two types of constraints: general constraints and specific constraints.

General constraints are:

C_0 – A teacher can only give one lecture at a time.

C_1 – A room can only host one lecture at a time.
C_2 – A student can only attend one lecture at a time.
C_3 – Room capacities must be respected.
C_4 – A few lectures are fixed.
C_5 – Teachers have days or hours of non availability.

Specific constraints are:

C_6 – Two lectures about a same subject must not be scheduled the same day.
C_7 – Lectures can only begin after 8 am and must end before 8 pm.
C_8 – A teacher may not teach more than 6 hours a day.
C_9 – Lunches need one and a half hours and must begin between 11.45 am and 12.45 am.
C_{10} – TDMs must not be scheduled in parallel with third year student's lectures.
C_{11} – TDMs of a same subject must be scheduled at the same time for the one-third groups, also TDEs of a same subject must be scheduled at the same time for the half-groups.
C_{12} – As far as possible, all lectures of the same group must be scheduled in a same room.
C_{13} – Students must have a 15 min pause between lectures.

91 lectures involving 42 teachers have to be scheduled. Those lectures can be scheduled in 8 different rooms and their starting time can take 240 values. Our time unit is a quarter of an hour (48 quarters in a day beginning at 8 am and ending at 8 pm, for a five day week). At the present time, the manual resolution of this timetabling problem requires several days of work.

4 A brief overview of the finite domains in CHIP

CHIP (Constraint Handling In Prolog) is a Constraint Logic Programming language from ECRC and now developed and sold by Cosytec [18, 24, 44]. The CHIP language can handle constraints over three distinct domains: finite domains, booleans and rationals. We briefly present the finite domains which were used for our application.

4.1 Numerical and symbolic constraints

Each constrained variable has a domain (set of scalar values) which must be declared *a priori*. CHIP provides the usual numerical constraints: equality (`#=`), dis-equality (`#\=`), inequalities (`#<`, `#<=`,`#>`,`#>=`). Each one can be applied over linear terms built upon domain variables and constants.

Symbolic constraints are also handled. A very useful one is `element(N, List, Value)`. It specifies, as an internal system constraint, that the `Nth` element of the list `List` must have the value `Value`. `List` is a non empty list of natural numbers and `Value` is either a natural number, a domain variable or a free variable.

The most interesting use of the constraint `element/3` is when `N` or `Value` are domain variables. Therefore as soon as the domains of `N` or `Value` change, a new constraint is dynamically added and inconsistent values are removed from the domains. The `atmost/3` (`atleast/3`) constraints state that at most (at least) `N` elements of a list `List` have the value `Value`.

The domain of the variables are reduced according to classical consistency techniques (Arc Consistency (AC)) and filtering algorithm (look ahead or partial look ahead).

4.2 The cumulative constraint

This constraint has been created [1] to solve scheduling problems difficult to solve with only the above presented constraints:

`cumulative([S1,...,Sn], [D1,...,Dn],[R1,...,Rn], L).`

where `[S1,...,Sn]`, `[D1,...,Dn]`, `[R1,...,Rn]` are non empty lists of domain variables and `L` is a natural number.

The usual interpretation of this constraint is a single resource scheduling problem. The S_is represent the starting times of the tasks, the D_is are the durations and the R_is the amount of resource needed by each task. L is the total amount of resource available at each instant of time. The cumulative constraint ensures that, at each instant of time i of the schedule, the consumed amount of resource cannot exceed the limit L. This is a simplified presentation of the cumulative constraint which owns in fact 8 arguments [18].

Let us consider three major uses of the cumulative constraint [1].

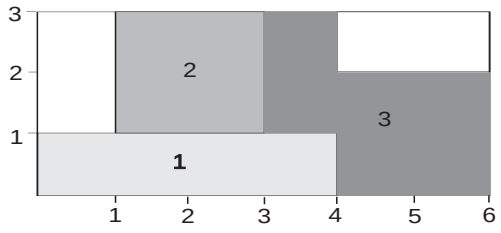

Fig. 1. cumulative([1,2,4],[4,2,3],[1,2,2],3)

1. Considering figure 1, there are three tasks to schedule: the first task uses one unit of the resource during four consecutive periods; tasks 2 and 3 use two units during respectively two and three periods. At any time the total amount of resource used by the different tasks is always less than or equal to 3.

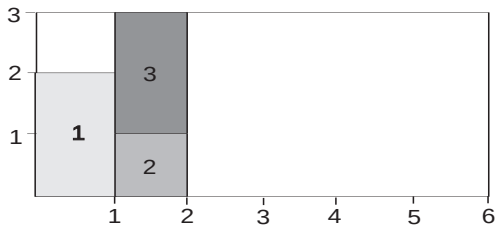

Fig. 2. cumulative([1,2,2],[1,1,1],[2,1,2],3)

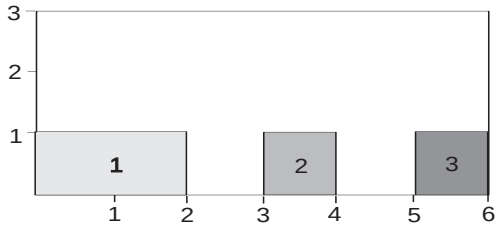

Fig. 3. cumulative([1,4,6],[2,1,1],[1,1,1],1)

2. For figure 2, all the tasks durations are equal to one. This particular case corresponds to the bin-packing problem [40]: m bins of fixed capacities and n objects of fixed size to put in these bins.
3. The third example (see figure 3) does not allow a cumulative amount of resource greater than 1. This corresponds in scheduling to the problem of tasks that cannot be executed at the same time because they share the same resource, the so called disjunctive tasks.

5 Solving our timetabling problem

The usual development of an application over finite domains consists of modeling the data and the domain variables, thus imposing the constraints, and finally defining the labeling strategy. We follow this pattern in our presentation.

5.1 Data representation

The domain variables are for each lecture: `Room`, `Day`, `Hour` and `Qh` (Quarter of an Hour), variables which domains are as follow: `Room :: 1..8`, `Day :: 1..5`, `Hour :: 1..48`, `Qh :: 1..240` (the 5 days are decomposed in 5×48 quarters).

The hour of beginning of each lecture is then represented twice. The choice between these representations depends on their fitting to the expression of the

corresponding constraint. The link between them is maintained by the constraint: `Qh = 48*(Day-1) + Hour`. To model breaks (a quarter of an hour) between lectures, we have chosen to *include* them at the beginning of each lecture. So, the length of each lecture is increased of a quarter of an hour.

5.2 Constraints modeling

In this section, we describe our problem in terms of CSP using the built-in constraints provided by CHIP.

The problem consists of 90 lectures and 500 constraints (110 of them are cumulative constraints).

Some constraints can be expressed directly:

C_{13} is taken into account in the data.
C_4 and C_{11} are expressed by an equality constraint.
C_5 and C_7 are expressed by domains restrictions.
C_9 is expressed by creating virtual lectures for lunches.

The other constraints are expressed by the symbolic constraints of CHIP, and especially by the cumulative constraint:

C_0 for each teacher, we set the disjunction of all his lectures by the cumulative constraint. Let `ListQh` and `ListLength` be respectively the list of the starting quarters of the lectures given by a teacher, and the list of the lengths of these lectures. The constraint is written:

```
cumulative(ListQh,ListLength,ListOf1,1)
```

where `ListOf1` is a list with the same length that `ListQh` but only composed of 1s. Thus, we consider each lecture as a task to perform with a resource (the teacher) whose capacity is of course one.

C_1, C_2, C_{10} The modeling of C_1, C_2 and C_{10} is based on the same principle as the one of C_0 by considering that each lecture is a task whose execution needs one unit of a resource of capacity 1. In C_1, this resource is a room, in C_2, a group of students and in C_{10} a student.

C_6 constraint claims, by the `alldifferent/1` predicate, that two lectures on a same subject cannot occur the same day.

C_8 is also expressed by the cumulative constraint: for each teacher, the length of the lectures he gives and the `Day` variables associated are collected into two lists. The constraint is then given by: `cumulative(ListOfDays, ListOf1, ListOfLength, R)`. The teacher is considered as a resource of capacity `R=4*6` hours and the lectures are the elementary tasks that consume an amount of resource equal to their length.

C_3 is introduced by the `element/3` constraint. Let `Room` be the domain variable of a lecture and `ListCap` the list containing the rooms capacities. The constraint is then: `element(Room,ListCap,Cap), Cap #>= size of the group`.

C_{12} is taken into account during the labeling by the `indomain2/2` predicate which tries to assign a lecture into the room assigned to its group.

Notice that for C_0, C_1, C_2 and C_{10}, the cumulative constraint expresses disjunctive constraints but for C_8 it's a capacity constraint.

5.3 Labeling

We tried several enumeration strategies. We only present four of them to illustrate our work. The first two are basic strategies in the CSP community. The third and fourth, inherited from OR, apply the first-fit decreasing principle, a well known method for solving bin-packing problems [40].

naive1 : lectures are selected by the `first_fail` principle. The values of the variables `Day` and `Hour` are determined by the `indomain/1` predicate which chooses the smallest value of their domain (*i.e.* the lectures are assigned as early as possible in the week).

naive2 : at each step, the lecture chosen is the longest one (in case of equality, the `first_fail` principle is used). The values of the variables are selected by the `indomain/1` predicate.

first_fit1 : each lecture is selected by the `first_fail` principle. The day with the shortest length of lectures (for the corresponding group) is assigned to it. The hour is selected by the `indomain/1`.

first_fit2 : at each step, the lecture chosen is the longest one (in case of equality, the `first_fail` principle is used). The day and hour are selected as in `first_fit1`.

5.4 First results

The first two labeling strategies leave us without solution after 48 hours of computation time, whereas the last two first-fit strategies give us a solution in two seconds. This is due to the fact that a naive labeling places the lectures at the beginning of the week. Impossibilities appear then very late and cause a lot of useless backtracks. First-fit enumeration spreads lectures uniformly over the week. The resulting timetable is then well-balanced for the students.

The weekly timetable for the year 1993-1994 is computed in 2 seconds of `cpu` time thanks to the `first_fit2` enumeration. We tested the accuracy of our approach by lowering the length of a working day. Initially a working day begins at 8 am and ends at 8 pm. When we set the beginning time at 8.45 am and the finishing time at 7.30 pm, we still have solutions (in about 2–5 s - $\approx$ 100 backtracks). If we try to reduce the working day further, the program has not found any solution in less than 5 minutes ($>$ 30000 backtracks).

Those results show the importance of a user defined labeling in the CLP framework. The CSP community is also aware of this view point. As quoted by E. Tsang [43], *one important issue is the ordering in which the variables are labeled and how the orderings could affect the efficiency of the search strategies significantly. The OR community is also aware of this conclusion: we can simply think about a graph coloring problem for which exists an ordering of the vertices which allows a simple coloration method to perform good results.*

5.5 Extensions

We have tested our approach upon another institute in our University for which we had to schedule about sixty lectures (the constraints are similar). A first-fit labeling strategy finds solutions in similar computation times (2–5 s).

These two institutes share common lectures, so, we tried a global solving of the timetabling problem, i.e. 165 lectures in 11 different rooms. Our program does not find any solution after 2 days of computing time for this problem. But, we are able to rapidly produce solutions by manually relaxing a few constraints (2–3) about teachers' availabilities.

The person who really builds the timetables in the two institutes considers that there is no *feasible* solution for the global problem under all the previously specified constraints. When he builds the timetables, he has to overcome several types of constraints and manually relax them depending on the lectures or teachers involved.

6 Constraint Relaxation

We said before that the global timetabling problem has no solution, thus, it seems very important to include a constraints relaxation system in an automated timetabling system. The fact that a solution exists in case of the manual relaxation of certain constraints (as we said above) and probably no feasible solution for the complete problem exists, comforts us in our position. We will, in the following sections, briefly survey about constraint relaxation in a logic programming environment and we will present our works.

6.1 Basic concepts

The majority of the concepts about constraint relaxation in a PLC scheme were introduced in [9], and [29] proposed a first theoretical framework.

In constraint relaxation problems, one must introduce a sort of hierarchy upon constraints. A weight is then proposed for each constraint. This weight allows the setting of a partial order relation between constraints. The lower the value of the weight, the more important the constraint is. Thus, constraints with weight 0 are called mandatory constraints, and those with a strictly positive weight are called preferred constraints.

A substitution (values for variables) which satisfies all the required constraints and which satisfies the preferred constraints in the best possible way with respect to a given comparator is called a solution. Thus, we are searching a maximal (to a given criteria) sub problem of the initial problem. This sub problem must have solutions.

To achieve constraint relaxation, one must answer three main questions:

- **when** exactly during the computation do we have to use constraint relaxation,

- **which** constraint do we need to relax in order to obtain a consistent sub problem,
- **how** do we have to perform the relaxation of the chosen constraint.

6.2 Former answers

As far as we know, no system is able to efficiently answer all the questions we raised above. In the literature, we identified two systems which fully answered the three questions but they were too simple for the kind of problems we want to solve [9, 33]. The others only answer one or two questions. In in [19, 35] the *who* is answered using responsibility between constraint or between variables to determine a *good* constraint to relax. In [19] the *when* ins asnwered using global consistency techniques. Nevertheless, those systems show good ways of works. Thus, it seems quite evident that before using a constraint relaxation module, we have to identify an inconsistency in the constraints system. Moreover, to choose the constraint that we have to relax to obtain a solution, we may try to identify the constraints responsible for the inconsistency.

Finally, we found a way to answer the *how* question in the ideas proposed in the dynamic CSP community [5, 6, 34, 37]. Let us recall that a dynamic CSP is a CSP in which constraints can be added or retracted during the computation. Besides, we think that the incremental system to interpret Prolog from [45] seems very interesting.

More global approaches were proposed in [25, 28]. Those approaches transform the original constraint relaxation problem into an optimization problem. This new problem is then solved with a classical branch and bound technique. We think this approach does not allow the intervention of the user during the process and thus seems unsatisfactory.

6.3 Our works

The aim of our work is to propose a complete constraint relaxation system. We started with a system which tried to make some of the above described propositions work together. We obtained a system which, when a Prolog fail occurred, used a constraint relaxation module using the identification techniques presented in [35] (The search of a dependence graph between constraints). This system, named FELIAC, has been implemented [8] above an incremental system which used ideas from [45].

We tried then to define a more unified system to solve the constraint relaxation problem. Thus, we worked on the ATMS (Assumption-based Truth Maintenance Systems) philosophy [27] to obtain a new system.

This new system associates a *justification* with each removal of a value from a variable during the computation. This justification will allow an easy identification of the causes of the removal of the value. Thus, in the case of an inconsistency, the recorded justifications lead to the identification of constraints responsible for the failed computation. We can then resume the computation

erasing the effects of the removed constraint (to achieve this, we just have to state that the justifications depending on this constraint are no longer valid). We must add that all the preferred constraints are considered as ATMS *suppositions* and then can be added or removed easily. So the ATMS approach can provide a complete constraint relaxation system. We are currently working on more improvements in our new system.

7 Conclusion

Our experience shows how timetabling problems can be efficiently solved by the CLP approach. Our experience highlights the CLP scheme capability of combining both CSP and OR approaches. As shown in this paper, one can declaratively state the problem in terms of constraints and define efficient search strategies inherited from OR works. This declarative capability needs high level constraints. Defining such constraints is only possible by importing OR works and techniques in CLP [2, 3, 16, 36]. This illustrates the huge capacity of prototyping an implementation of real-life applications in constraint logic programming [38, 39]. Moreover, the conciseness of the programs and the short development times leads us to rapidly develop alternative versions. Indeed, various heuristics have been developed, tested and validated in a very short development time.

Our experience points out the importance of introducing Constraint Relaxation in CLP. Indeed, various timetabling problems (and other real-life problems) can be found to be over-constrained. The development facility of the CLP paradigm is not sufficient, it has to be extended for Constraint Relaxation. Thus, we are currently working on the embedding of Constraint Relaxation into CLP languages. At the moment, we are implementing our new general model based upon ATMS and validating it using timetabling problems.

References

1. Abderrahmane Aggoun and Nicolas Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.
2. P. Baptiste, C. Le Pape, and W. Nuijten. Incorporating efficient Operations Research algorithms in constraint-based scheduling. In 1^{st} *Joint Workshop on Artificial Intelligence and Operational Research*, 1995. To appear.
3. N. Beldiceanu and E. Contjean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.
4. Jacques Bellone, André Chamard, and C. Pradelles. Plane : An evolutive system for aircraft production written in CHIP. In *Proceedings of the First International Conference on Practical Applications of Prolog*, London, 1992.
5. Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.
6. Christian Bessière. Arc consistency for non-binary dynamic CSPs. In *Proceedings ECAI'92*, 1992.

7. Patrice Boizumault, Yann Delon, and Laurent Péridy. Constraint logic programming for examination timetabling. To appear in Journal of Logic Programming, 1995.
8. Patrice Boizumault, Christelle Guéret, and Narendra Jussien. Efficient labeling and constraint relaxation for solving time tabling problems. In Pierre Lim and Jean Jourdan, editors, *Proceedings of the 1994 ILPS post-conference workshop on Constraint Languages/Systems and their use in Problem Modeling : Volume 1 (Applications and Modelling)*, Technical Report ECRC-94-38, ECRC, Munich, Germany, November 1994.
9. Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In Giorgio Levi and Maurizio Martelli, editors, *ICLP'89: Proceedings 6th International Conference on Logic Programming*, pages 149–164, Lisbon, Portugal, June 1989. MIT Press.
10. J. P. Boufflet and S. Negre. About planning an examination session. In *Proceedings ECCO VII, Conference of the European Chapter on Combinatorial Optimization*, Milan, Italy, February 1994.
11. J. P. Boufflet and S. Negre. A practical timetable problem. In *EURO XII, Operations Research Designing Practical Solutions*, Glasgow, United Kingdom, July 1994.
12. Bull SA. *AMPHI+ user's guide*, 1992.
13. Edmund Burke, David Elliman, and Rupert Weare. A Genetic Algorithm for university timetabling. In *AISB Workshop on Evolutionary Computing*, University of Leeds, UK, April 1994.
14. M. Cangalovié and J.A.M. Schreuder. Exact algorithm fro weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operations Research*, 1991.
15. M. W. Carter. A survey of practical applications of examination timetabling algorithms. *European Journal of Operations Research*, 34(2), 1986.
16. Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In 11^{th} *International Conference on Logic Programming*, 1994.
17. André Chamard, Frédéric Decès, and Annie Fischler. Applying CHIP to a complex scheduling problem. In *JICSLP'92*, Washington, DC, November 1992. (Submitted).
18. Cosytec SA. *CHIP v4 reference manual*, 1993.
19. Xavier Cousin. *Application de la programmation logique avec contraintes au problème d'emploi du temps*. PhD thesis, Université de Rennes I, France, 1993. In French.
20. Dominique de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.
21. Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving a Cutting-Stock Problem in Constraint Logic Programming. In Robert Kowalski and Kenneth Bowen, editors, *ICLP'88: Proceedings 5th International Conference on Logic Programming*, pages 42–58, Seattle, WA, August 1988. MIT Press.
22. Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the Car Sequencing Problem in Constraint Logic Programming. In *ECAI-88: European Conference on Artificial Intelligence*, Munich, August 1988.
23. Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, 8(1-2):74–94, January-March 1990.

24. Mehmet Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *FGCS-88: Proceedings International Conference on Fifth Generation Computer Systems*, pages 693–702, Tokyo, December 1988. ICOT.
25. François Fages, Julian Fowler, and Thierry Sola. Handling preferences in constraint logic programming with relational optimization. In *PLILP'94*, Madrid, September 1994.
26. R. Feldman and M.C. Golumbic. Optimization algorithms for student scheduling via constraint satisfiability. *The Computer Journal*, 33(4), 1990.
27. Kenneth D. Forbus and Johan de Kleer. *Building Problem Solvers.* MIT Press, Cambridge, 1993.
28. Julian Fowler. Preferred constraints as optimization. In *Proceedings Journées Françaises de Programmation en Logique*, 1993.
29. Eugene Freuder. Partial constraint satisfaction. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 278–283, Detroit, 1989.
30. F. Glover. Tabu search. CAAI Report 88–3, University of Colorado, 1988.
31. F. Glover. Tabu search: a tutorial. *Interfaces*, 20:74–94, 1990.
32. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
33. Michael Jampel and David Gilbert. Fair Hierarchical Constraint Logic Programming. In Manfred Meyer, editor, *Proceedings ECAI'94 Workshop on Constraint Processing*, Amsterdam, August 1994.
34. Philippe Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution - Propagation de contraintes dans les réseaux dynamiques.* Thèse de doctorat, Université des Sciences et Techniques du Languedoc, Montpellier, France, January 1991. In French.
35. Francisco Menezes, Pedro Barahona, and Philippe Codognet. An incremental hierarchical constraint solver. In Paris Kanellakis, Jean-Louis Lassez, and Vijay Saraswat, editors, *PPCP'93: First Workshop on Principles and Practice of Constraint Programming*, Providence RI, 1993.
36. C. Le Pape. Programmation par contraintes : une nouvelle forme de débat entre recherche opérationnelle et intelligence artificielle. *Bulletin de l'AFIA*, (21), April 1995.
37. Projet CSPFlex., Gérard Bel, Éric Bensana, Khaled Ghédira, David Lesaint, Thomas Schiex, Gérard Verfaillie, Christine Gaspin, Roger Martin-Clouaire, Jean-Pierre Rellier, Pierre Berlandier, Bertrand Neveu, Brigitte Trousse, Héène Fargier, Jérôme Lang, Philippe David, Philippe Jansenn, Tibor Kökény, Marie-Catherine Vilarem, and Philippe Jégou. Représentation et traitement pratique de la flexibilité dans les problèmes sous contraintes. In *Actes des Journées Nationales du PRC GDR Intelligence Artificielle*, Marseille, France, October 1992. In French.
38. Michel Rueher. A first exploration of Prolog-III's capabilities. *Software – Practice and Experience*, 23, 1993.
39. Michel Rueher and Bruno Legeard. Which role for CLP in software engineering? An investigation on the basis of first applications. In *Proceedings of the First International Conference on Practical Applications of Prolog*, London, 1992.
40. Smith and Shing. Bin packing. *Bulletin of the IMA*, 19, 1983.
41. Arabinda Tripathy. A lagrangean relaxation approach to course timetabling. *Journal of Operational Research Society*, 31:599–603, 1980.

42. Arabinda Tripathy. School timetabling - a case in large binary integer linear programming. *Management Science*, 30(12), 1984.
43. Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
44. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.
45. Pascal Van Hentenryck. Incremental constraint satisfaction in logic programming. In *Proceedings* 6^{th} *International Conference on Logic Programming*, 1989.
46. G. M. White and L. Kang. A logic approach to the resolution of constraints in timetabling. *European Journal of Operations Research*, 61:306–317, 1992.
47. M. Yoshikawa, K. Kaneko, Y. Yomura, and M. Watanabe. A constraint-based approach to high-school timetabling problems: A case study. In *Proceedings of AAAI'94 Conference*, 1994.