

Efficient Labelling and Constraint Relaxation for Solving Time Tabling Problems

P. Boizumault¹, C. Guéret², N. Jussien,
email: boizu@info.emn.fr jussien@info.emn.fr

1. Introduction

Constraint Logic Programming over Finite Domains, CLP(FD), has been in constant development for a few years. CLP(FD) allows an efficient resolution of various highly combinatorial problems in scheduling, cutting-stock, warehouse location, planning, molecular biology ... [DIN 90a,b,c], [BAP 92], [BEL 92], [CHA 92], [CLA 93], [BOI 94b]. From a software engineering view point, this approach provides an easy prototyping and induces rapid development times. This is due to the integration of a powerful constraint propagation mechanism (Consistency Techniques) in a Logic Programming language (Prolog).

The aim of this paper is to show, through a practical application (solving time tabling problems), that modelling is no longer sufficient to solve a constraint based problem. It is even more important to label with good strategies and to be able to relax constraints. In fact, in a lot of practical problems, usual labelling strategies are not enough to deal with the underlying combinatorial aspect. We must then define heuristics which are mostly imported from Operations Research. Over-constrained problems are also frequent. With this kind of problems the user cannot be let without any answer. A constraint relaxation mechanism must be provided in order to obtain a sub-problem which has a solution.

We will first specify the time tabling problem for our institute and then describe and evaluate the various approaches used in Operations Research to solve these kinds of problems. Then we present the development in CHIP of an automatic time tabling tool for our institute, emphasising on the labelling stage. Finally, we will discuss about the coping of constraint relaxation with our application.

¹Ecole des Mines de Nantes, 3 rue Marcel Sembat, 44049 NANTES CEDEX.

²Institut de Mathématiques Appliquées, 3 place André Leroy, BP 808, 49008 ANGERS CEDEX 01.

2. Solving time tabling problems

General time tabling problem can be defined as the scheduling of a set of lectures to which must attend several groups of students, over a preset period of time, using some resources and satisfying a certain set of constraints. Many researchers [WER 85], [CAR 85] have dealt with this problem since the fifties. And nowadays this problem is always studied thanks to its variety and its complexity [BUL 92], [WHI 92]. In this section, we present the time tabling problem for our institute and the OR proposals to solve this kind of problems.

2.1. Our problem

The IMA (Insitute of Applied Mathematics) provides a five years formation after the French Baccalaureate. The first three years require a weekly time table whereas the last two are yearly organised. In order to simplify the presentation, we will only speak about the weekly time tables.

The first three years represent 160 students (60-50-50). Most of the teached subjects are cut in lectures, directed works given by teachers (called TDE) and directed works given by third year students (called TDM). A 4 hours period is reserved for examination. A few lectures are fixed independently of our institute. Complete groups attend to lectures, whereas doubled or tripled groups attend to TDE and TDM. At last, all lectures take place in 8 different rooms with various capacities.

2.2. Constraints

We can distinguish two types of constraints: general constraints and specific constraints for our institute.

General constraints are:

- C0* - A teacher can only give one lecture at a time.
- C1* - A room can only host one lecture at a time.
- C2* - A student can only attend one lecture at a time.
- C3* - Room capacities must be respected.
- C4* - A few lectures are fixed.
- C5* - Teachers have days or hours of non disponibility.

Specific constraints are:

- C6* - Two lectures about a same subject must not be scheduled the same day.
- C7* - Lectures can only begin after 8 am and must end before 8 p.m.
- C8* - A teacher may not teach more than 6 hours a day.
- C9* - Lunches need an hour and a half and must begin between 11.45 am and 12.45 am.
- C10* - TDMs must not be scheduled at the same of third year student's lectures.
- C11* - TDMs of a same subject must be scheduled at the same time.

C12 - As far as possible, all lectures of a same group must be scheduled in a same room.

C13 - Students must have a quarter pause between lectures.

91 lectures involving 42 teachers have to be scheduled. Those lectures can be scheduled in 8 different rooms and their starting date can take 240 different values. Our time unit is the quarter (48 quarters in a day beginning at 8 am and ending at 8 p.m., for a five days week). The number of solutions in the case of a complete and naive enumeration is $91^{(240*58)}$. Actually, the manual resolution of this time tabling problem requires several days of work.

2.3. O.R. Approaches

2.3.1. Graph Coloring

The time tabling problem is equivalent to a graph vertex coloring problem [WER 85], [CAR 85], [CaS 91], [Cou 93] where :

Each lecture (a triplet (class, teacher, room)) makes a vertex.

Two vertices are connected if the associate lectures have a student, a teacher or a room in common.

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

We cannot introduce easily all types of constraints (e.g. teachers impossibilities) in this modelisation. Further more, crucial information about the nature of the constraints are lost. Indeed, an edge can represent the fact that a student must attend the two lectures, as well as the fact that those two lectures must take place in the same room or that they have the same teacher.

2.3.2. Flow problems

Edge graph coloring problems can be resolved with heuristics about compatible flow search problems ([Cou 93], [dWE 85]). Such a method is interesting because we know efficient algorithms for solving that type of problems. But, in the case of solving a time tabling problem over T periods of time we must find T flows, one for each quarter in our a case. This problem is now again NP-Complete. Further more, we cannot introduce all types of constraints (e.g. precedence constraints). At last, this method requires same duration for all lectures, which seems not very realistic.

2.3.3. Mathematical Programming

A third method that can be used is mathematical programming with integers ([Cou 93], [WER 85]). The main problem in this case is the number of data and

constraints. Optimal resolution of an integer linear program is NP - Hard in the general case, except if the constraint matrix is, for example, totally unimodular (not so often in real life).

In order to reduce the problem size, a redefinition of the variables by grouping students, rooms or lectures is possible. A. Tripathy proposes such a transformation in [TRI 80] and [TRI 84]. He uses next a Langragean Relaxation method. However, his method (slightly complicated), in spite of certain reduction of the problem complexity, only applies to the precise problem he wants to solve. If we add constraints or modify the problem, we must reconsider the entire analysis.

2.3.4. Genetic Algorithms

In analogy with the reproduction of living beings, the process starts with an initial population of random solutions[FAN 94]. Each solution is coded with a chromosome: a vector of symbols which length is $2N$ (number of lectures), divided in N contiguous pieces containing two genes. The two genes of i^{th} piece represent the hour of beginning and the room of the lecture number i . A fitness function, defined from the number and the type of constraints which do not hold, is used to measure the quality of a solution. In each generation, a pair of reproductive chromosomes is randomly chosen, with a certain probability which increases with their fitness. Each chosen pair (x,y) is used in crossing-over : a gene is randomly determined in x and exchanged with the gene at the same position in y . The new population is then the former one plus the new chromosomes from the mutation and the crossing over. The number of solutions is kept constant by the elimination of the one with the smaller fitness value. Genetic algorithms may find efficient solutions. However, all the parameters must be determined with experimentation. Further more, they do not have any warranty of convergence.

So, there is a large scale of time tabling problems. In each case, specific constraints arise. OR classical methods do not seem very efficient when directly applied because of the variety of constraints to deal with. Indeed, very often, these methods cannot give a good model for the whole problem we face and we have then to relax it (some constraints are suppressed).

3. A brief overview of CHIP

CHIP (Constraint Handling In Prolog) is a Constraint Logic Programming language from ECRC and now developed and commercialised by Cosytec [DHS 88], [VHE 89a] [COS 93]. The CHIP language can handle constraints over three distinct domains: finite domains, booleans, rationals. We briefly present the finite domains which were used for our application.

3.1. 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 (\neq), inequalities ($<$, \leq , $>$, \geq). Each one can be applied over linear terms built upon domain variables and constants.

CHIP provides various symbolic constraints :

`element(N, List, Value)` the N th element of the list `List` has the value `Value`.

`alldifferent(L)` all the elements of the list `L` are different two by two.

`atmost(N, List, Value)` the list `List` contains at most N elements with the value `Value`.

`atleast(N, List, Value)` the list `List` contains at least N elements with the value `Value`.

3.2. The cumulative constraint

This constraint has been created [AGB 92] in order to solve scheduling problems difficult to handle with only the constraints presented above :

`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 natural number.

The usual interpretation of this constraint is a scheduling single resource problem. The `Si` represent the date of beginning of the tasks, the `Di` are the duration times and the `Ri` the amount of resource needed by each task. `L` is the amount of resource allowed for sharing at each instant of time. The cumulative constraint means that, at each 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 [COS 93].

4. Solving our time tabling problem

We will first present our modelling of the constraints in CHIP (an intensive use of the well suited cumulative constraint is done). Then we will show the importance of an efficient labelling strategy in order to obtain reasonable computation times for large-sized problems.

4.1. Data representation

A lecture is represented by a functional term : `c(year, group, size, teacher, subject, length, Room, Day, Hour, Qh)` where `year`, `group`, `size`, `teacher`, `subject` and `length` are known. We want to determine `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 $5 \cdot 48$ quarters)

The hour of beginning of each lecture is then doubly represented. 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.

4.2. Constraints modelling

In this section, we show the adequacy of CPL(FD) languages for easily modelling the constraints inherent to our time tabling problem.

Some constraints can be expressed directly :

C13 is taken into account in the datas.

C4 and *C11* are expressed by an equality constraint.

C5 and *C7* are expressed by domains restrictions.

C9 is expressed by creating virtual lectures for lunches.

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

C0 : 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 quarter of the lectures given by a teacher, and the list of the length 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 with 1. Thus, we consider each lecture as a task to perform with a resource (the teacher) which capacity is of course one.

The modelling of *C1*, *C2* and *C10* is based on the same principle as the one of *C0* by considering that each lecture is a task whose execution needs one unit of a resource of capacity 1. In *C1*, this resource is a room, in *C2*, a group of students and in *C10* a student.

C6 constraint claims, by the `alldifferent` predicate, that two lectures on a same subject do not take place the same day.

C8 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.

C3 is introduced by the `element/3` constraint. Let `c(year, group, size, teacher, subject, length, Room, Day, Hour, Qh)` be a lecture and *ListCap* the list containing the rooms capacities. The constraint is then : `element(Room, ListCap, Cap), Cap #>= size of the group`.

C12 is taken into account during the labelling stage by the `indomain2/2` predicate. This predicate tries to affect a lecture in the room affected to its group.

4.3. Labelling

We implemented several labelling strategies:

`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 affected 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 and the day with the smallest duration of lectures (for the corresponding group) is affected to it.

`first_fit2` : at each step, the lecture chosen is the longest one (in case of equality, the `first_fail` principle is used); the day with the smallest duration of lectures (for the corresponding group) is affected to it.

4.4. First results

The first two labelling strategies let 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 labelling places the lectures at the beginning of the week. Impossibilities appear then very late and provoke a lot of useless backtracks. `First_fit` enumeration spreads uniformly lectures over the week. The resulting time table is then well-balanced for the students.

The weekly time table for the year 1993-1994 can be settled in 2s thanks to the `first_fit` enumeration. We tested the adequacy of our approach by lowering the length of a working day. Initially a working day begins at 8am and ends at 8pm. When we set the beginning hour at 8.45am and the ending hour at 7.30pm, we always have solutions (in about 2-5s - \approx 100 backtracks). If we try to reduce more the working day, the program does not find any solution in less than 5 minutes (>30000 backtracks).

Those results show the importance of a user defined labelling in the CLP(FD) framework. The CSP community is also conscient of this view point. As quoted by E. Tsang [TSA 93], one important issue is the ordering in which the variables are labelled and in which 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.

4.5. Extensions

We have tested our approach upon another institute in our university for which we had to schedule about sixty lectures (the constraints are rather the same). A first_fit labelling strategy finds solutions in similar computation times (2-5 s).

These two institutes share common lectures, so, we tried a global solving of the time tabling problem, i.e. 165 lectures in 11 different rooms. The total number of potential solutions in a complete and naive enumeration is $165^{(240*11)}$. 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 constraints about teachers' disponibilities.

The person who actually manually builds the time tables for 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.

5. Constraints relaxation

For implementing a realistic time tabling software, relaxing constraints is dominating. We will, in the following sections, make a brief survey about constraint relaxation in the CLP framework³ and we will present our current works.

5.1. Basic concepts

The major part of the concepts about constraint relaxation in the CLP scheme were introduced in [BMW 89] and [FRE 89] who proposed a theoretical framework.

In constraints relaxation problem, 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 is the more important the constraint is. Thus, constraints with weight 0 are called required 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

³We can find a more global vision of constraint relaxation in [YKN 94], but this vision is clearly outside the CLP framework.

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.

5.2. Former works

In [BMW 89], constraints are taken in account level by level (a level of constraints is built with all the constraints with the same weight), until the resolution of the resulting problem fails. This method is comparable to an "automatic experimentation" according to [Cou 93].

A more global view of the resolution is presented in [FOW 93]. Finding a solution to a problem with preferred constraint is seen as an optimisation problem. This is achieved by applying negation, subsumption and branch and bound rules.

Regardless to the CLP control, a more interesting approach is handled by methods which deal with the constraints in the order of their appearance. They have to choose one or several constraints whose relaxation will suppress a failing condition, and effectively suppress the chosen constraint(s) from the constraints system.

For choosing the constraints to relax, [Cou 93] and [MBC 93] propose methods based upon intelligent backtracking to select constraints responsible for a failure during a computation (this may be an immediate responsible [Cou 93] or a responsible by secondary effects [MBC 93]). These two methods use a dependence graph. The analysis of such a graph allows an efficient choice of constraints whose relaxation may make possible the determination of a solution for the problem.

The suppression of a constraint is achieved by recomputing a solution from the instant t_i preceding the introduction of the constraint we want to suppress. But, by using this approach, we do not use what we learned between the instant t_i and the current instant t of the computation (when we found the failure). It would be interesting to find a suppression method which uses all the already done work. [COU 93] and [MBC 93] do not propose such a method. Thus, it seems interesting to look at incremental constraints system as proposed in [FMB 90] and [VHE 89b].

5.3. Our works

P. Van Hentenryck [VHE 89b] proposes a new technique called reexecution which recomputes from the moment preceding the introduction of the constraint we want to suppress. Let P be a problem defined by a set of constraints and Q be a problem deduced from P by adding a constraint. We can say that what is not a solution for P is also not a solution for Q . Thus, all what we have rejected during the resolution of the problem P can be rejected during the resolution of the problem Q .

P. Van Hentenryck proposes to record, in what he calls an oracle, information about what has been rejected during the resolution of P . These information may be clauses in standard Prolog resolution, or chosen values and resulting domains during a labelling in a CLP resolution... An intensive use of these information gives better results in terms of computation time.

We based our work upon the excellent detection methods proposed by [MBC 93] and upon the incremental system proposed by [VHE 89b] and [FMB 90] in order to improve the computing efficiency. We built a prototype, the FELIAC system, joining efficient constraint detection and incremental resolution methods.

The FELIAC system invokes the constraint detection method as it arises the first Prolog fail. The FELIAC system is built with an incremental CLP system using Van Hentenryck techniques.

We experimented our system. To make comparisons accurate, we developed a version of the system proposed by [BMW 89] and we implemented IHCS(FD) from [MBC 93]. We present here results about a problem presented in [Cou 93] : Michel, Paul and Alain must attend to working sessions taking place over 4 half days. Paul and Alain want to expose their work to Michel and Michel then wants to expose his work to Paul and Alain. Each presentation takes one half day. We can define several required constraints : someone who attends a presentation cannot present something in the same half day. Two different persons cannot present to a same third person at the same time. We can also define preferred constraints : Michel wants to know what Paul and Alain have done before presenting his work (weight 1). Michel would like not to come on the fourth half day (weight 3). Michel does not want to present his work to Alain and Paul at the same time (weight 2). There is no solution for the problem respecting all the constraints.

Here are the first results :

<i>Method</i>	<i>Number of relaxed constraints</i>	<i>Number of backtracks</i>
HCLP [BMW 89]	1	12
IHCS [MBC 93]	3	3
FELIAC	1	3

Joining an efficient constraint detection and a constraint satisfaction incremental system allows the calculus of better solutions according to the number of relaxed constraints and to the number of backtracks. We are now testing our approach on the above presented time tabling problem. Unfortunately, we cannot exhibit actual results about this application because our modelling makes an intense use of the cumulative constraint and FELIAC can currently only handle linear numerical constraints.

6. Conclusions and further works

Our experience demonstrates that the CLP(FD) programmer needs high level primitives to express its particular constraints. Such primitives must be efficiently implemented in order to reduce drastically the search space.

Our experience shows the importance of an intelligent labelling strategy. In fact, if constraint propagation techniques are able to reduce the search space, the developer must define intelligent labelling strategies in order to deal with the underlying combinatorial aspect. Such techniques need works closely related to OR and CSP approaches.

Our experience points out the importance of introducing the relaxation of constraints in CLP. Most real life problems (time tabling by example) induce imperative constraints and preference constraints. But with the CLP(FD) tools available today, we cannot efficiently tackle constraints relaxation.

Finally our experience illustrates the huge capacity of prototyping and implementation of real-life applications in constraint logic programming [RUE 92, 93]. Moreover, The conciseness of the programs and the short development times lead us to develop rapidly alternative versions. Indeed, various heuristics have been developed, tested and validated in a very short development time.

We are currently investigating two directions. First, we are defining a graphical interactive system which helps the time tabling manager by automatically propagating constraints and detecting incompatibilities [BOU 92]. Secondly, we continue to develop the FELIAC system in order to take into account symbolic constraints relaxation. For this, we are investigating dynamic CSP [JEG 91], [SYN 92].

7. Bibliography

- [AGB 92] A. Aggoun, N. Beldiceanu, Extending CHIP in order to solve Complex Scheduling Placement Problems, Journées Francophones de Programmation en Logique, Lille, 1992.
- [BAP 92] P. Baptiste, B. Legeard, C. Varnier, Hoist Scheduling Problem: an approach based on Constraint Programming, IEEE Int. Conference on Robotics and Automation, Nice, 1992.
- [BEL 92] J. Bellone, A. Chamard, C. Pradelles, Plane: An evolutive System for Aircraft Production written in CHIP, First Int. Conference on Practical Applications of Prolog, London, 1992.
- [BOU 92] J.P. Boufflet, Emplois du temps dans un environnement fortement contraints: exemple de l'UTC, Thèse de l'Université Technologique de Compiègne, 1992.
- [BMW 89] A. Borning, M. Maher, A. Martindale, M. Wilson, Constraint Hierarchies and Logic Programming, in proceedings of the 6th International Conference on Logic Programming, ICLP'86, Lisboa 1989.
- [BOI 94a] P. Boizumault, Y. Delon, L. Péridy, Vers une approche mixte PLC-RO: un premier pas au travers d'une application, RFIA'94, Paris, 1994.
- [BOI 94b] P. Boizumault, Y. Delon, L. Péridy, Planning exams using CLP, 2nd ICPAP'94, London, 1994.

- [BUL 92] AMPHI+, user guide, BULL SA 1992.
- [CAR 86] Michael W.Carter, A survey of practical applications of examination timetabling algorithms, *Operations Research* 34 n°2, 1986.
- [CaS 91] M. Cangalović, J.A.M. Schreuder, Exact algorithm for weighted graphs applied to timetabling problems with lectures of different lengths., *European Journal of Operational Research*, 1991.
- [CHA 92] A. Chamard, F. Decès, A. Fischler, Applications du langage CHIP à un problème complexe d'ordonnancement, *JFPL*, Lille, 1992.
- [CLA 93] D. Clark, C. Rawlings, J. Shirazi, L. Li, K. Schuerman, M. Reeve, A. Veron, Solving Large Combinatorial Problems in Molecular Biology Using the ElipSys Parallel Constraint Logic Programming System, *The Computer Journal*, vol 36, n°8, 1993.
- [Cou 93] X. Cousin, Application de la Programmation en Logique avec Contraintes au problème d'Emploi du Temps, Thèse de l'Université de Rennes, Juin 1993.
- [COS 93] CHIP v4 reference manual, Cosytec SA, may 1993.
- [DIN 90a] M. Dincbas, H. Simonis, P. Van Hentenrick. Solving Cutting-Stock Problem in Constraint Logic Programming. Fifth International Logic Programming Conference Seattle, 1990.
- [DIN 90b] M. Dincbas, H. Simonis, P. Van Hentenrick. Solving a Car-Sequencing Problem in Constraint Logic Programming. Fifth Int. Logic Programming Conference Seattle, 1990.
- [DIN 90c] M. Dincbas, H. Simonis, P. Van Hentenrick. Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming* - 8, pages 75-93, 1990.
- [DHS 88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf , F. Berthier, The Constraint Logic Programming Language CHIP, in *FGCS'88*, 693-702, Tokyo, 1988.
- [FAN 94] H.L. Fang, D. Corne, P. Ross, Successful Lecture Timetabling using Genetic Algorithms, *ECAI 94*.
- [FMB 90] B. N. Freeman-Benson, J. Maloney, A. Borning, An Incremental Constraint Solver, *CACM* 33(1), 1990.
- [FOW 93] J. Fowler, Preferred Constraints as Optimisation, *JFPL'93*, Nimes, 1993.
- [FRE 89] E. Freuder, Partial Constraint Satisfaction, *IJCAI'89*, 1989.
- [HAR 80] R. Haralick, G. Elliot, Increasing tree search efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, 14:263-313, 1980.
- [LeP 91] J.P. Le Pape, D. Ranson, CLEF or programming with constraints, logic, equations and functions, 4th Int. Conf. on Software Engineering and its applications, Toulouse, 1991.
- [JEG 91] P. 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 de Montpellier II, Janvier 1991.

- [MBC 93] F. Menezes, P. Barahona, P. Codognet, An incremental hierarchical constraint solver applied to a time-tabling problem, Avignon 1993.
- [RUE 92] M. Rueher, B. Legeard, Which role for CLP in Software Engineering? An investigation on the basis of first applications, First International Conference on Practical Applications of Prolog, London, 1992.
- [RUE 93] M. Rueher, A first exploration of PrologIII's capabilities, Software-Practice and Experience, vol. 23(2), pp177-200, 1993.
- [SYN 92] G. Bel, E. Bensana, K. Ghedira, D. Lesaint, T. Schiex, G. Verfaillie, C. Gaspin, R. Martin-Clouaire, J.P. Rellier, P. Berlandier, B. Neveu, B. Trousse, H. Fargier, J. Lang, P. David, P. Janssen, T. Kökeny, M.C. Vilarem, P. Jegou, Représentation et traitement pratique de la flexibilité dans les problèmes sous contraintes, Journées Nationales PRC IA, Marseille, 1992.
- [TRI 80] Arabinda Tripathy, A Lagrangean Relaxation Approach to Course Timetabling, J. OpI. Soc, Vol 31, pp599-603, 1980.
- [TRI 84] Arabinda Tripathy, School timetabling - a case in large binary integer linear programming, Management science, Vol 30, N°12, dec 1984.
- [TSA 93] E. Tsang, Foundation of Constraints Satisfaction, Academic Press, 1993.
- [VHE 89a] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, Cambridge, 1989.
- [VHE 89b] P. Van Hentenryck, Incremental Constraint Satisfaction in Logic Programming, ICLP'6, Lisboa, 1989.
- [WER 85] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19, 1985.
- [WHI 92] G. M. White, L. Kang, A Logic Approach to the Resolution of Constraints in Timetabling, European Journal of Operations Research, vol. 61, pp. 306-317, 1992.
- [YKN 94] M. Yoshikawa, K. Kaneko, Y. Nomura, M. Watanabe, A Constraint-Based Approach to High-School Timetabling Problems : A Case Study, Proceedings of the AAAI'94 conference, 1994.