

# Dynamic Backtracking with Constraint Propagation

## Application to numeric CSPs

Narendra Jussien and Olivier Lhomme  
École des Mines de Nantes – Département Informatique  
4 Rue Alfred Kastler – BP 20722  
F-44307 Nantes Cedex 3 – France

### Abstract

Recent works on constraint relaxation [Jussien, 1997; Jussien and Boizumault, 1997b] provided the DECORUM system (Deduction-based Constraint Relaxation Management). That system can be seen as an integration of arc-consistency within the *dynamic backtracking* algorithm [Ginsberg, 1993]. *Dynamic backtracking replaces* the backtracking process by a much less blind behavior that consists in local modifications of the choices made up to the current situation. In this paper, a new enumeration technique over numeric CSPs is proposed: *dynamic domain splitting*. This is a domain splitting search where chronological backtracking is replaced by a kind of *dynamic backtracking*.

## 1 Introduction

Many industrial and engineering problems can be seen as constraint satisfaction problems (CSPs). A CSP is defined by a set of variables each with an associated domain of possible values and a set of constraints on the variables. This paper deals with CSPs where the constraints are numeric relations and where the domains are either finite integer domains or continuous domains (numeric CSPs). Numeric CSPs can be used to express a large number of problems, in particular physical models involving imprecise data or partially defined parameters.

In general, numeric CSPs cannot be tackled with computer algebra systems, and most numeric algorithms cannot guarantee correctness. The only numeric algorithms that guarantee correctness – even when floating-point computations are used – are either coming from the interval analysis community or from the AI community (CSP).

Some works in solving such numeric CSPs can be considered as “adaptations” to numeric constraints of symbolic CSP ideas. More precisely, they generally define a new partial consistency that can be computed by an associated filtering algorithm with a good efficiency (with regard to the domain reductions performed).

For example, [Lhomme, 1993; 1994; Haroud and Faltings, 1994] aim at defining concepts of higher order consistencies similar to  $k$ -consistency [Freuder, 1978]. [Faltings, 1994; Faltings and Gelle, 1997] propose in a sense to merge the

constraints concerning the same variables, giving one “total” constraint (thanks to numerical analysis techniques) and perform arc-consistency on the total constraints. [Benhamou *et al.*, 1994; Hentenryck *et al.*, 1997] aim at expressing interval analysis prunings as partial consistencies.

All those techniques have as common point to be incomplete, and then require to be interleaved with an enumeration process (typically a dichotomic one). Constraint-solving algorithms are then a search-tree exploration where at each node is applied one of the filtering techniques above. When a node leads to a contradiction, a chronological backtracking is performed.

What is proposed in this paper is yet another adaptation of a CSP idea to numeric CSPs. It does not consist in another filtering technique but rather in a new search tree exploration. To the authors’ knowledge, no work aiming at improving that search-tree exploration has been done on numeric CSPs, although chronological backtracking could be evicted by several candidates. For example:

- *Conflict-directed backjumping* [Prosser, 1993] is able to identify the reasons of a failure and thus to backtrack in the search tree higher than chronological backtracking.
- *dynamic backtracking* proposed in [Ginsberg, 1993] aims at replacing backtracking (intelligent or not) by the suppression of the cause of the failure *without loss of the work made between the cause and the failure*.

Constraint propagation has been tentatively introduced in those algorithms: *conflict-directed backjumping* has been improved with forward-checking [Prosser, 1993] and arc-consistency [Prosser, 1995], *dynamic backtracking* has been improved with forward-checking [Verfaillie and Schiex, 1994].

In this paper, we will use a more recent work [Jussien, 1997; Jussien and Boizumault, 1997a; 1997b] that in some sense, generalizes *dynamic backtracking* plus arc-consistency for static CSPs and dynamic CSPs. They propose a general scheme –Relax(D,C,P)– and a system –DECORUM– to relax constraints in a dynamic context. Two features of that work does import here:

- The first one, whose roots can be tracked from the DNAC4 algorithm [Bessière, 1991], makes filtering techniques over CSP incremental for constraint deletion and addition.
- The second feature, automatic identification of the constraints to relax, allows the enumeration process to be seen as a sequence of equality constraint addition and deletion. That is, as in dynamic backtracking, there is no longer backtracking after a failure: it is replaced by a “surgical operation” on the causes of the failure. The incrementality of the filtering algorithm makes that enumeration process efficient.

[Jussien and Boizumault, 1997c] explains how this system can be seen as dynamic backtracking plus arc-consistency for symbolic static or dynamic CSPs. In this paper, we show how the idea of *dynamic backtracking* can be useful for solving numeric CSPs, leading to an intelligent search tree exploration over the static numeric CSPs. Yet in fact, dynamic numeric CSPs can also be handled

easily by minor modifications. But this paper primarily aims at showing that a dynamic constraint solver, being designed to reuse as possible the past search tree exploration, can be of great interest also for static CSPs.

The paper is organised as follows. In section 2, we recall the *dynamic backtracking* algorithm. Section 3 presents the DECORUM system's basic machinery. Section 4 summarizes how DECORUM can be considered as *dynamic backtracking* plus arc-consistency, and section 5 presents *dynamic domain splitting*, a new search-tree exploration for numeric CSPs.

## 2 Dynamic backtracking

Enumerations algorithms for CSPs are mainly based upon chronological backtracking. The key idea to improve such algorithms is to *learn* from failure. Learning is achieved through the notion of *nogood* borrowed from the TMS community [Doyle, 1979]. A *nogood* is defined as a *globally inconsistent partial assignment* [Verfaillie and Schiex, 1995]. More precisely, a partial assignment  $A$  is a nogood if and only if no solution contains  $A$ .

A first improvement of chronological backtracking has been provided through Prosser's [1993] *conflict-directed backjumping* (CBJ) algorithm. This algorithm implicitly uses nogoods by determining relevant backtrack points<sup>1</sup> without losing any solution.

*Dynamic backtracking* [Ginsberg, 1993] goes further. Back jumping is interesting but a lot of possibly useful and independant work is irremediably lost upon backtracking. Nogoods should be used at the maximum of their possibilities. *dynamic backtracking* records and uses them through the notion of *eliminating explanation*. Let  $N$  be a nogood and  $v$  the variable in  $N$  chosen as backtrack point.  $N \setminus \{v = a\}$  is an eliminating explanation for the value  $a$  assigned to variable  $v$  in  $N$ . The key idea of *dynamic backtracking* relies in not performing any backtrack: variables chosen as *backtrack point* are unassigned without modifying any other assignment. That search processes by *jumps*. The completeness of the approach is ensured by the set of eliminating explanations.

*Dynamic backtracking* thus improves chronological backtracking in two ways:

- it saves effort by identifying *good backtrack points*.
- it avoids thrashing (recomputing of already explored sub-parts of the search space) by making *surgical* changes when a contradiction occurs.

The main drawback of *dynamic backtracking* is that it does not make use of the constraints to guide the search – no reduction is performed after any assignment.

## 3 The DECORUM system

The DECORUM system handles over-constrained problems arising in dynamic environments [Jussien, 1997; Jussien and Boizumault, 1997a; 1997b]. Those

---

<sup>1</sup>Apart from the last assigned variable.

works provide a general schema for handling them and DECORUM is an implementation of the presented ideas for CSPs.

DECORUM relies upon two main considerations:

- When a contradiction occurs in a dynamic problem, a constraint to relax must be chosen taking into account both its real effects on the variables of the problem<sup>2</sup>, and the user’s preferences on the constraints<sup>3</sup>.
- When relaxing a constraint, a complete reexecution must be avoided as much as possible. For that, information on the constraint past effects must be kept in order to ensure a real incrementality. The roots of this point can be tracked from the DNAC4 algorithm [Bessière, 1991] which enforces arc-consistency in a constraints set after the removal or addition of one constraint.

In order to efficiently handle CSPs, filtering algorithms are usually used. Such an algorithm performs *domain reductions*: values – or tuples of values – are removed because they do not appear in any solution of the considered problem.

As told above, information about past effects of constraints need to be kept. This can be done by asking the filtering algorithm to give an *explanation* for each domain reduction it performs. Informally, an *explanation* for the removal of value  $a$  from the domain of variable  $v$  is a subset  $E$  of constraints having no solution together with the assignment of  $a$  to  $v$ .

The core of DECORUM is an arc-consistency filtering algorithm that provides explanations. It will be called an ACE algorithm (arc-consistency with explanations) throughout the paper. Such an algorithm is designed from any classical arc-consistency filtering algorithm: either a static one (such as AC4 [Mohr and Henderson, 1986] or AC5 [Van Hentenryck *et al.*, 1992]) or a dynamic one (such as DNAC4 [Bessière, 1991] or DNAC6 [Debruyne, 1995]).

The need to relax a constraint seems to elect a dynamic version of an ACE algorithm. Yet, in fact maintaining explanations for the domain reductions trivially makes dynamic any static algorithm: it is sufficient to forget reductions whose explanations contains the constraint to relax and to repropagate the constraints upon the corresponding variable. The basic functionalities of an ACE algorithm are then twofolds:

- propagation of a new constraint in order to achieve arc-consistency:  
function `addConstraint(c,explanation)`.  
The second parameter is an explanation for the constraint addition.
- “depropagation” of a constraint:  
function `removeConstraint(setOfConstraints)`.  
The past effects of the constraints are deleted and arc-consistency is achieved.

Furthermore, when a contradiction occurs during the computation of arc-consistency, the ACE algorithm gives an explanation for it.

---

<sup>2</sup>It must be partly responsible for the contradiction.

<sup>3</sup>Some constraints are more important than others.

**Providing explanations** Explanations are computed by the ACE algorithm. Let us consider an example: an AC5-based ACE algorithm. The AC5 algorithm actively uses the semantics of the handled constraints. For example, when filtering a constraint  $x > y$  considering that the current domain of  $x$  is  $\{\dots, a\}$ , values greater than  $a$  will be removed from  $d_y$  and an *explanation* for these removals can be  $E = \bigcup_{b \in d_x, b > a} E_{x \neq b}$  where  $E_\delta$  represents an explanation for reduction  $\delta$ .

**Performing efficient constraint removal** Past effects of a constraint must be erased upon its removal. For a constraint  $c$ , let us define the set  $\alpha(c)$  as the set of all the explanations containing constraint  $c$ . If the constraint  $c$  is removed, all these explanations are no more valid *i.e.* the associated value removal may be reconsidered.

Operationnally, the associated values of the set  $\alpha(c)$  are put back in their respective domain<sup>4</sup>. In order to ensure arc-consistency, these reintroduced values need to be tested for another removal explanation. If such an explanation exists, the value is removed as usual.

## 4 Dynamic backtracking + arc-consistency

Arc-consistency has already been merged with the standard backtracking algorithm leading to the MAC algorithm (Maintaining Arc-Consistency) [Sabin and Freuder, 1994] which is one of the best solving algorithm for CSP. The idea is to propagate constraints in order to achieve arc-consistency after each instantiation. What about integration of constraint propagation with other enumeration algorithms?

*Conflict-directed backjumping* [Prosser, 1993] is able to identify the reasons of a failure and thus to backtrack in the search tree higher than chronological backtracking. To derive benefit from constraint propagation, Prosser also proposed an hybridation of *conflict-directed backjumping* with *Forward Checking*: namely the FC-CBJ algorithm. FC-CBJ performs forward checking reduction upon instantiation. Nevertheless, Bessière and Régin [Bessière and Régin, 1996] showed that the MAC algorithm [Sabin and Freuder, 1994], based on chronological backtracking, was far better than FC-CBJ on hard problems. Arc-consistency also has been introduced within *intelligent backtracking* algorithms [Prosser, 1995]. Unfortunately, the overhead due to the modification of the propagation algorithm in order to provide explanations was not compensated by the improvements in the search.

Schiex et Verfaillie have proposed an integration of dynamic backtracking with forward checking for static [Verfaillie and Schiex, 1995] and dynamic CSPs [Verfaillie and Schiex, 1994]. But, [Verfaillie and Schiex, 1994] have observed that forward checking was apparently much more beneficial to *intelligent backtracking* algorithms than to dynamic backtracking.

---

<sup>4</sup>In a DNAC4-like algorithm the set  $\alpha(c)$  is not explicitly represented, but is computed from the *justif* structure. The *justif* structure associates to each value removal the constraint origin of the deletion

People therefore thought there was no need in integrating arc-consistency within improvements of standard backtracking (e.g. [Bessière and Régin, 1996] concluded that using “*intelligent*” backtracking techniques is useless). And thus, no proposition have been done in order to improve *dynamic backtracking* with arc-consistency.

The propagation step is easy to integrate in *dynamic backtracking*: after each assignment. But, when unassigning a variable (leaving untouched all the other variables), it is necessary to be able to undo only the effects of the reconsidered assignment without reconsidering the effects of the propagation of the others assignments *i.e.* the algorithm must be able to achieve arc-consistency as if the reconsidered assignment had never been made and without a complete reexecution in order to keep good performances.

In fact, an ACE algorithm, as presented in the previous section, provides all the necessary functionalities to integrate arc-consistency within *dynamic backtracking*. The key idea is to consider an assignment as the addition of an equality constraint between a variable and its value and to consider an unassignment as a constraint removal. Contradiction explanations (made up of enumeration constraints) are usable within the *dynamic backtracking* algorithm as eliminating explanation. The ACE algorithm can be used in order to: propagate constraints, remove constraints and provide contradiction explanations. More details can be seeked in [Jussien and Boizumault, 1997c].

In the following, we present a *dynamic backtracking*-based enumeration algorithm for solving numeric CSPs.

## 5 Dynamic backtracking-like enumeration over numeric CSPs

### 5.1 Dynamic Bound-Consistency for Intervals

The numeric CSPs fundamental idea was popularized by [Davis, 1987]: the projection of a numeric constraint over the axes of its variables is computed with interval arithmetic. For instance, the interval propagation algorithm is a Waltz algorithm using that idea.

Depending on whether holes in a domain are allowed or not, the partial consistency ensured by a filtering algorithm concerns all the values (like arc-consistency) or only the bounds of the domains (like k-B-consistency [Lhomme, 1993], where B stands for bounds).

For simplicity, we will assume here that holes in a domain are not allowed. Thus, for each domain only two values are to be kept: its lower bound and its upper bound. So we will only consider bound-consistency.

When considering constraint removal, explanations must be kept for each interval modification to be able to determine which set of values are to be put back into the considered domain. The notion of explanation for each value removal (*cf.* section 3) can be adapted here to explanation for removal of a set of values (either from the left of the interval or from the right). Indeed, only

two explanations<sup>5</sup> are to be kept for each domain in order to provide further explanations: how the lower bound has been reached and how the upper bound has been reached.

In the same way we defined an ACE algorithm from any AC algorithm, we can define a BCE algorithm (*bound-consistency with explanations*) from a bound-consistency filtering algorithm. Basically, we use the same machinery but with a different basis: a bound-consistency algorithm. Note that, due to bound-consistency, when putting back value, a single interval need to be computed *i.e.* the convex hull of the resulting interval is answered whilst performing the necessary modifications to the explanations system.

## 5.2 Dynamic domain splitting

The typical way of finding solutions over numeric CSPs is to perform *domain splitting*, a dichotomic enumeration interleaved with a filtering technique: the filtering technique is applied on the numeric CSP, the domain of a variable is split in two, and the two resulting numeric CSPs are explored separately by chronological backtracking.

That enumeration process can be seen as a sequence of additions or suppressions of splitting constraints, which are inequality constraints.

The *dynamic domain splitting* algorithm proposed here (*cf.* figure 1) proceeds as *dynamic backtracking*: each time a failure occurs during the search, the cause of the failure is deleted without complete loss of the work made between the cause and the failure. The enumeration process is here a dichotomic one. A splitting constraint in the *dynamic domain splitting* algorithm plays the role of an assignment in *dynamic backtracking*. Thus, a variable may have several splitting constraints at the same time.

There are two kinds of splitting constraint addition:

- the first ones are due to a new splitting. Choosing the first part of the variable domain is performed by the addition of a constraint to the constraint system.
- The second ones are due to an already active splitting. A splitting is a choice between two splitting constraints  $c_1$  and  $c_2$  (one for each subpart of the domain). We say  $c_1$  ( $c_2$ ) is the negation splitting constraint of  $c_2$  ( $c_1$ ), and we note  $c_1 = \neg c_2$ . If a splitting constraint  $c$  is removed and if its negation splitting constraint  $\neg c$  has not been tried (it has no eliminating explanation), then  $\neg c$  has to be added.

An explanation is here a set of splitting constraints. As soon a contradiction is detected, the contradiction explanation is analysed and its most recent splitting constraint is deleted (with all its effects). All we have to do is, when removing the effects of a constraint  $c$ , to remove all the “descendants” of the constraint, *i.e.* the splitting constraints on the same domain that are subsequent to  $c$  (that point is not mandatory since the subsequent splitting constraints on

---

<sup>5</sup>Domain reductions are based only on the current value of the considered intervals.

the same domain either would lead to a contradiction or would be subsumed by the next splitting constraint addition.)

Finally it remains to explain what to do after a removal of a splitting constraint when its negation splitting constraint (corresponding to the other part of the domain) has an eliminating explanation. In chronological backtracking, this would mean that one has to backtrack higher in the search tree. Here this is done by raising a contradiction, with as explanation the union of the eliminating explanations of the two splitting constraints. This will lead to another splitting constraint removal.

### 5.3 Implementation sketch

The main loop consists first in choosing a variable to split; it stops when there is no more such variable (for instance all the domains are sufficiently tight). Then a splitting constraint is added to the constraint system to choose the first part of the domain<sup>6</sup>.

When a failure is detected (i.e. a domain becomes empty), the function *removeMostRecentSplit* is called with the contradiction explanation as parameter. If the contradiction explanation is an empty set, that means the constraint system has no solution. Else, the most recent splitting constraint  $c$  is removed with all its effects. As explained in the previous section, all the effects of the “descendants” of the constraint have also to be removed. Then either the negation splitting constraint of  $c$  has an eliminating explanation or not:

- In the first case, the eliminating explanations of the two splitting constraint removal are unioned. Then what we have to do is only to recursively call the function *removeMostRecentSplit* with that parameter.
- In the second case, we keep the eliminating explanation of  $c$  which is the contradiction explanation minus the constraint (see section 2). Then the negation splitting constraint that corresponds to the other part of a domain is added.

Completeness and termination can be shown as completeness and termination of *dynamic backtracking*.

### 5.4 Usefulness of the approach

Consider a problem with independant subproblems  $P_1$  and  $P_2$ , and an enumeration order that interleaves the enumeration of the variable of the subproblems. For example,

1. some variables of  $P_1$  are splitted ( $A$ ),
2. then all the variables of  $P_2$  are splitted until  $P_2$  is solved ( $B$ ),
3. finally the remaining variables of  $P_1$  are splitted ( $C$ ).

---

<sup>6</sup>for simplicity, we will assume in this paper that the first part of the domain is always the left subinterval.

```

procedure dynamicDomainSplitting
(1) begin
(2)   while there exists a variable  $i$  to split do
(3)      $c \leftarrow i \in \text{left of } D_i$  % new splitting constraint
(4)     if not addConstraint( $c, \{c\}$ ) then
(5)        $E \leftarrow$  returned contradiction explanation
(6)       removeMostRecentSplit( $E$ )
(7)     endif
(8)   endwhile
(9) endwhile
(10) end

procedure removeMostRecentSplit(in  $E$ )
(11) begin
(12)   if  $E = \emptyset$  then
(13)     failure
(14)   else
(15)      $c \leftarrow$  the most recent splitting constraint in  $E$ 
(16)      $S \leftarrow \{c\} \cup$  all the splitting constraints descendant of  $c$ 
(17)     if  $\neg c$  has an eliminating explanation then
(18)        $lastTry \leftarrow true$ 
(19)        $explPreviousTry \leftarrow$  eliminating explanation of  $\neg c$ 
(20)     else
(21)        $lastTry \leftarrow false$ 
(22)     endif
(23)     delete all the eliminating explanations of splitting constraints containing an element of  $S$ 
(24)     if not removeConstraint( $S$ ) then
(25)        $E' \leftarrow$  contradiction explanation
(26)       removeMostRecentSplit( $E'$ )
(27)     endif
(28)     if  $E \setminus \{c\}$  is valid then
(29)       if  $lastTry$  then
(30)         removeMostRecentSplit( $explPreviousTry \cup (E \setminus \{c\})$ )
(31)       else
(32)         keep  $E \setminus \{c\}$  as eliminating explanation for  $c$ 
(33)         if not addConstraint( $\neg c, \{\neg c\}$ ) then
(34)            $E' \leftarrow$  contradiction explanation
(35)           removeMostRecentSplit( $E'$ )
(36)         endif
(37)       endif
(38)     endif
(39)   endif
(40) end

```

Figure 1: Dynamic Domain Splitting

Domain splitting, performing chronological backtracking, will lose all the work done to solve  $P_2$  at each time it needs to backtrack from  $C$  to  $A$ , and worse, all the solution of  $P_2$  will be enumerated. In continuous domains, the solution set is not always a discrete set, then you can imagine why sometimes the continuous solvers do not give answers...

Unlike domain splitting, *dynamic domain splitting* will not change the domains of the variables in  $P_2$  when removing a splitting constraint from  $(A)$ . The reason is there cannot be a splitting constraint over a variable of  $P_2$  in an explanation of contradiction of a splitting constraint over a variable of  $P_1$  because of their independence.

Our very first experiments have shown such a behavior over several constraint systems, leading to drastic improvement in performance. Nevertheless, when this kind of thrashing does not occur, there may exist a substantial overhead in time. Experiments need to be done on a more large scale.

## 6 Conclusion and perspectives

This paper has presented a new enumeration algorithm for solving numeric static CSPs. First experiments are promising, and experiments over standard benchmarks over numeric CSPs are being done to compare *dynamic domain splitting* with more classical approaches.

That algorithm may be easily modified in order to handle dynamic numeric CSPs. But this paper primarily aims at showing how ideas from dynamic CSPs can be of great interest also for static CSPs.

## References

- [Benhamou *et al.*, 1994] F. Benhamou, D. Mc Allester, and P. Van Hentenryck. CLP(Intervals revisited). In *International Logic Programming Symposium*. MIT Press, 1994.
- [Bessière and Régin, 1996] Christian Bessière and Jean-Charles Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problem. In *CP'96*, Cambridge, MA, 1996.
- [Bessière, 1991] Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.
- [Davis, 1987] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(2):281–331, 1987.
- [Debruyne, 1995] Romuald Debruyne. Les algorithmes d'arc-consistance dans les CSP dynamiques. *Revue d'Intelligence Artificielle*, 9(3), 1995.
- [Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.
- [Faltings and Gelle, 1997] B. Faltings and Esther Gelle. Local consistency for ternary numeric constraints. In *IJCAI*, August 1997.

- [Faltings, 1994] B. Faltings. Arc consistency for continuous variables. *Artificial Intelligence*, 65(2), 1994.
- [Freuder, 1978] Eugene Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, November 1978.
- [Ginsberg, 1993] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [Haroud and Faltings, 1994] D. Haroud and B. Faltings. Global consistency for continuous constraints. In Alan Borning, editor, *Second Workshop on Principles and Practice of Constraint Programming*, Seattle, May 1994.
- [Hentenryck *et al.*, 1997] P. Van Hentenryck, D. Mc Allester, and D. Kapur. Solving polynomial systems using branch and prune approach. *SIAM Journal*, 34(2), 1997.
- [Jussien and Boizumault, 1997a] Narendra Jussien and Patrice Boizumault. A best first approach for solving over-constrained dynamic problems. In *IJCAI'97*, Nagoya, Japan, August 1997. (poster – RR 97-6-INFO – École des Mines de Nantes).
- [Jussien and Boizumault, 1997b] Narendra Jussien and Patrice Boizumault. Best-first search for property maintenance in reactive constraints systems. In *International Logic Programming Symposium*, pages 339–353, Port Jefferson, N.Y., oct 1997. MIT Press.
- [Jussien and Boizumault, 1997c] Narendra Jussien and Patrice Boizumault. Dynamic backtracking with constraint propagation – application to static and dynamic CSPs. In *CP97 Workshop on the theory and practice of dynamic constraint satisfaction*, 1997. submitted.
- [Jussien, 1997] Narendra Jussien. *Relaxation de Contraintes pour les problèmes dynamiques*. PhD thesis, Université de Rennes I, 24 October 1997.
- [Lhomme, 1993] O. Lhomme. Consistency techniques for numeric CSPs. In *IJCAI'93*, pages 232–238, Chambéry, France, August 1993.
- [Lhomme, 1994] O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles*. PhD thesis, Université de Nice – Sophia Antipolis BP 145 06903 Sophia Antipolis, 1994.
- [Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
- [Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, August 1993. (Also available as Technical Report AISL-46-91, Strathclyde, 1991).
- [Prosser, 1995] Patrick Prosser. MAC-CBJ: maintaining arc-consistency with conflict-directed backjumping. Research Report 95/177, Department of Computer Science – University of Strathclyde, 1995.

- [Sabin and Freuder, 1994] Daniel Sabin and Eugene Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).
- [Van Hentenryck *et al.*, 1992] Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2–3):291–321, October 1992.
- [Verfaillie and Schiex, 1994] Gérard Verfaillie and Thomas Schiex. Dynamic backtracking for dynamic csps. In Thomas Schiex and Christian Bessière, editors, *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, Amsterdam, August 1994.
- [Verfaillie and Schiex, 1995] Gérard Verfaillie and Thomas Schiex. Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. *Revue d'Intelligence Artificielle*, 9(3), 1995.