

Dynamic Backtracking with Constraint Propagation

Narendra Jussien, Olivier Lhomme and Patrice Boizumault
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). In this paper, we show how the ideas developed in that system can be used in order to integrate Constraint Propagation 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. Thus the whole constraint programming community may derive benefits from its integration with a constraint propagation algorithm.

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 classic CSPs where the domains are finite sets of values and with numeric CSPs where the constraints are numeric relations and where the domains are either finite integer domains or continuous domains (numeric CSPs).

Such problems are solved using enumeration techniques based mainly upon standard backtracking. Standard backtrack presents several drawbacks that have been handled through various improvements leading to Intelligent Backtracking techniques (*eg.* Conflict-directed BackJumping [Prosser, 1993]). More recently, [Ginsberg, 1993] presented the dynamic backtracking algorithm which *replaces* the backtracking process by local modifications of the choices made up to the current situation.

On one hand, [Sabin and Freuder, 1994] showed that constraint propagation was well worth integrating with the enumeration process. On the other hand, [Bessière and Régin, 1996] showed that constraint propagation was not well worth integrating within conflict-directed backjumping cutting down interests in this field. What about integrating constraint propagation with dynamic backtracking? This question has not been answered yet because the integration is not as trivial as for standard backtracking and conflict-directed backjumping.

Recent works on constraint relaxation [Jussien, 1997; Jussien and Boizumault, 1997b] provided the DECORUM system (Deduction-based Constraint Relaxation Management). In this paper, we show how the ideas developed in that system can be of great use in order to integrate Constraint Propagation within the Dynamic Backtracking schema. Furthermore, we present the adaptation of

Journées du GDR Programmation. 12, 13, 14 novembre 1997. Orléans.

those ideas for numeric CSPs leading to a new search-tree exploration in that sub-field.

The paper is organized as follows: section 2 introduces the DECORUM system. In section 3, we recall the basics of dynamic Backtracking and section 4 presents how to integrate constraint propagation within dynamic backtracking for discrete CSPs. Finally, section 5 presents the modifications to be made in order to provide the same to numeric CSPs.

2 The DECORUM system

Recently, we presented the DECORUM system in order to handle over-constrained problems arising in dynamic environments [Jussien, 1997; Jussien and Boizumault, 1997a; 1997b]. Those 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 problem¹, and the user's preferences on the constraints².
- 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.

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`. Note that, if a contradiction occurs during the computation of arc-consistency, the ACE algorithm gives an explanation for it. An explanation for the constraint addition can be given upon a call to this function (*cf.* [Jussien, 1997]).
- “depropagation” of a constraint: function `removeConstraint`. The past effects of the constraints are deleted and arc-consistency is achieved.

¹It must be partly responsible for the contradiction.

²Some constraints are more important than others.

3 From standard backtracking to dynamic backtracking

Dynamic Backtracking [Ginsberg, 1993] is an enumeration algorithm which improves standard backtracking. We will present it through successive refinements of the standard backtracking technique.

Let V be a set of variables to be instantiated. Let \bar{V} be the set of instantiated variables. Let i, j be variables and a a value in the domain of i . Let D_i be the current domain of variable i . Let $\epsilon(i)$ be the set of values for variable i which are not compatible³ with the other already assigned variables.

Figure 1 recalls the *Standard Backtracking* algorithm. Upon a contradiction, this algorithm reconsiders the last choice done. Note that each unassignment must be done by resetting the domains of the variables to their state prior the assignment of the considered variable.

Standard backtracking can be improved by considering this simple statement: *the last choice point may not be relevant*. Indeed, it may be possible that the contradiction is due to a past choice made much earlier. Let us assume that there exists a function that provides an *explanation* for a contradiction *i.e.* a set of assigned variables whose joint assignments leads to the contradiction. Considering such a function leads to an intelligent backtracking algorithm (*cf.* figure 1). We will not precise here the way of implementing such an explanation mechanism. Note that each sequence of unassignments must include the resetting of the domains of the variables to their state prior to the oldest reconsidered assignment.

<pre> procedure standardBacktracking (1) begin (2) $\bar{V} \leftarrow \emptyset$ (3) while $V \neq \emptyset$ do (4) select i in V (5) $Values \leftarrow D_i \setminus \epsilon(i)$ (6) if $Values = \emptyset$ then (7) let j be the last variable in \bar{V} (8) unassign j (9) else (10) select a in $Values$ (11) assign a to i (12) $V \leftarrow V \setminus \{i\}$ (13) $\bar{V} \leftarrow \bar{V} \cup \{i\}$ (14) endif (15) endwhile (16) end </pre>	<pre> procedure intelligentBacktracking (1) begin (2) $\bar{V} \leftarrow \emptyset$ (3) while $V \neq \emptyset$ do (4) select i in V (5) $Values \leftarrow D_i \setminus \epsilon(i)$ (6) if $Values = \emptyset$ then (7) let E be an explanation of the contradiction (8) let j be the most recent variable in E (9) unassign all the variables assigned since j (10) else (11) select a in $Values$ (12) assign a to i (13) $V \leftarrow V \setminus \{i\}$ (14) $\bar{V} \leftarrow \bar{V} \cup \{i\}$ (15) endif (16) endwhile (17) end </pre>
--	---

Figure 1: Standard Backtracking and Intelligent Backtracking

Intelligent backtracking avoid the exploration of useless alternatives in the search tree but lacks of avoiding *thrashing*: upon backtracking possibly reusable parts of the undone computation are lost inducing a recomputation of already inferred information. The idea of the *Dynamic Backtracking* algorithm [Ginsberg, 1993] is to keep as much as possible already done search. That is, no backtrack is really performed, only local unassignments are done. In order to keep the completeness of the search, Ginsberg introduces the notion of *eliminating explanation* which gives the context of the unassignment (the remainder of the contradiction explanation). Those *eliminating explanations* ensure the completeness and the finiteness of the approach (*cf.* [Ginsberg, 1993] for proofs).

Figure 2 shows the *Dynamic Backtracking* algorithm.

³At least one constraint is not verified.

```

procedure dynamicBacktracking
(1)  begin
(2)     $\bar{V} \leftarrow \emptyset$ 
(3)    while  $V \neq \emptyset$  do
(4)      select  $i$  in  $V$ 
(5)       $Values \leftarrow D_i \setminus \epsilon(i)$ 
(6)      if  $Values = \emptyset$  then
(7)        let  $E$  be an explanation of the contradiction
(8)        if  $E = \emptyset$  then
(9)          failure
(10)       else
(11)         let  $j$  be the most recent variable in  $E$ 
(12)         unassign  $j$  with eliminating explanation  $E \setminus \{j\}$ 
(13)         remove all the explanations involving  $j$ 
(14)       endif
(15)     else
(16)       select  $a$  in  $Values$ 
(17)       assign  $a$  to  $i$ 
(18)        $V \leftarrow V \setminus \{i\}$ 
(19)        $\bar{V} \leftarrow \bar{V} \cup \{i\}$ 
(20)     endif
(21)   endwhile
(22) end

```

Figure 2: Dynamic Backtracking

4 Constraint propagation in Dynamic Backtracking for discrete CSP

Constraint propagation for discrete CSP 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 (*i.e.* to achieve arc-consistency) after each instantiation.

No proposition have been done in order to improve *Dynamic Backtracking* with constraint propagation. This is due to three main reasons:

- The propagation step is easy 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 have never been made and without a complete reexecution in order to keep good performances.
- It is necessary to modify the explanation system used in *Dynamic Backtracking* in order to take into account the propagation. This involves modifications of the arc-consistency algorithm.
- Constraint propagation has been tentatively 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. People therefore thought there was no need in integrating Constraint Propagation within improvements of standard backtracking (*cf.* [Bessière and Régin, 1996]).

In fact, as we saw in the previous section, the DECORUM system provides all the necessary tools to be able to integrate constraint propagation 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.

With this in mind, contradiction explanations (based upon constraints) are usable within the *Dynamic Backtracking* algorithm. Thus, the ACE algorithm used in DECORUM can be used in order to: propagate constraints, remove constraints and provide contradiction explanations.

Figure 3 gives a first version of the resulting algorithm. This first naive version does not work because integrating constraint propagation have many subtle effects on the behavior of that algorithm:

1. In line 5, there is no need to compute the set $\epsilon(i)$ because it is always empty due to the use of constraint propagation which removes non supported values.
2. The constraint removal on line 12 needs to be propagated using the DECORUM primitive `removeConstraint`. This constraint removal can lead to a contradiction. It is then necessary to relax constraints (unassign variables) until a consistent state is reached.
3. In line 14, there is a value removal. In order to propagate it, we add a *negative* constraint (variable \neq value) with an explanation. That behavior can possibly lead to a contradiction. Thus, the propagation step must then be achieved until consistency is reached (relaxing constraints as is done between line 7 and 14) or a failure is identified (the contradiction explanation is empty).
4. If the propagations of points 2 and 3 are achieved, the condition of line 6 will never be reached, since the contradictions will be captured in the propagation of value removals presented in point 3.

Taking into account all those points leads to the final version of the algorithm given also in figure 3. In this algorithm, modifications of sets V and \bar{V} are done when adding and removing constraints and therefore do not explicitly appear here. In this algorithm, point 1 is taken into account line 5, point 2 and 3 are taken into account through the procedure `unassignMostRecent`, and point 4 is taken into account in the new organization of the algorithm.

More details can be found in the DECORUM related papers [Jussien and Boizumault, 1997a; 1997b; 1997c].

5 Constraint propagation in Dynamic Backtracking for Numeric CSPs

This section focuses on CSPs where the constraints are numeric relations and where the domains are either finite integer domains or continuous domains (numeric CSPs).

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).

Those constraint-solving algorithms are typically a search-tree exploration (most of the time a dichotomic enumeration process) where a pruning (or filtering) technique is applied at each node. The existing improvements of this kind of algorithm can be categorized in the following way:

- Tighter pruning techniques – stronger consistencies.
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

```

procedure dynamicBacktrackingWithConstraintPropagation
– first version –
(1) begin
(2)    $\bar{V} \leftarrow \emptyset$ 
(3)   while  $V \neq \emptyset$  do
(4)     select  $i$  in  $V$ 
(5)      $Values \leftarrow D_i \setminus \epsilon(i)$ 
(6)     if  $Values = \emptyset$  then
(7)       let  $E$  be a contradiction explanation
(8)       if  $E = \emptyset$  then
(9)         failure
(10)      else
(11)        let  $j$  be the most recent variable in  $E$ 
(12)        remove constraint( $j = v_j$ )
(13)        remove all explanations involving  $j$ 
(14)        remove  $v_j$  from  $j$  because  $E \setminus \{j\}$ 
(15)      endif
(16)    else
(17)      select  $a$  in  $Values$ 
(18)      if adding constraint  $i = a$  is contradictory then
(19)        set  $i \neq a$  using the contradiction explanation
(20)      else
(21)         $V \leftarrow V \setminus \{i\}$ 
(22)         $\bar{V} \leftarrow \bar{V} \cup \{i\}$ 
(23)      endif
(24)    endif
(25)  endwhile
(26) end

```

```

procedure dynamicBacktrackingWithConstraintPropagation
– final version –
(1) begin
(2)    $\bar{V} \leftarrow \emptyset$ 
(3)   while  $V \neq \emptyset$  do
(4)     select  $i$  in  $V$ 
(5)     select  $v_i$  in  $D_i$ 
(6)     if not addConstraint( $i = v_i$ ) then
(7)        $E \leftarrow$  returned contradiction explanation
(8)       unassignMostRecent( $E$ )
(9)     endif
(10)  endwhile
(11) end

procedure unassignMostRecent(in  $E$ )
(12) begin
(13)   if  $E = \emptyset$  then
(14)     failure
(15)   else
(16)      $j \leftarrow$  the most recent variable in  $E$ 
(17)     if not removeConstraint( $j = v_j$ ) then
(18)        $E' \leftarrow$  contradiction explanation
(19)       unassignMostRecent( $E'$ )
(20)     endif
(21)     if  $E \setminus \{j\}$  is valid then
(22)       if not addConstraint( $j \neq v_j, E \setminus \{j\}$ ) then
(23)          $E' \leftarrow$  contradiction explanation
(24)         unassignMostRecent( $E'$ )
(25)       endif
(26)     endif
(27)   endif
(28) end

```

Figure 3: Dynamic Backtracking with Constraint Propagation – first and final versions

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

- Faster pruning techniques – better algorithms for a given consistency. [Lhomme *et al.*, 1996; 1998] aim at boosting interval propagation (and stronger consistency filtering algorithms built on top of it) by a dynamic analysis and optimization of its running.

The search-tree exploration on numeric CSPs is always performed by chronological backtracking. This section shows how the ideas presented in the previous sections allow a search tree exploration *à la Dynamic Backtracking* to be interleaved with pruning techniques. To the authors’ knowledge, this is the first attempt of improving the search-tree exploration for solving numeric CSPs.

5.1 Bound Consistency with explanations

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 explanations⁴ 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.

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 4) 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.

In addition to the basic filtering algorithm, which is a BCE algorithm in place of a ACE algorithm, there is another difference with the algorithm in figure 3. It is due to the enumeration process which is here a dichotomic one. A splitting constraint in the dynamic domain splitting algorithm

⁴Domain reductions are based only on the current value of the considered intervals.

```

procedure dynamicDomainSplitting
(1) begin
(2)   while there exists a variable  $i$  to split do
(3)     if not addConstraint( $i \in$  left of  $D_i$ ) then
(4)        $E \leftarrow$  returned contradiction explanation
(5)       unassignMostRecent( $E$ )
(6)     endif
(7)   endwhile
(8) endwhile
(9) end

```

Figure 4: Dynamic Domain Splitting

plays the role of an assignment in dynamic backtracking and of an equality constraint in dynamic backtracking with constraint propagation. Thus, a variable may have several splitting constraints at the same time. That is why we do not keep explanations as set of variables but as set of splitting constraints. 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 the same domain either would lead to a contradiction or would be subsumed by the next splitting constraint addition). Note that lines 17 and 22 in `unassignMostRecent` figure 3 need to be modified in order to take into account the specificity of splitting constraints. Therefore, calls to `removeConstraint` and `addConstraint` are modified in order to read c_j (splitting constraint on j) or $\neg c_j$ ⁵ instead of $j = v_j$ or $j \neq v_j$.

Proofs for completeness and termination are similar to those of dynamic backtracking.

5.3 Usefulness of the approach

Consider a problem with independent 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 split (A),
2. then all the variables of P_2 are split until P_2 is solved (B),
3. finally the remaining variables of P_1 are split (C).

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.

The same point may be illustrated for finite discrete problems (CSP) leading to the same interest although not as crucial since domains are finite.

⁵ $\neg c_j$ denotes the negation constraint of a splitting constraint c_j (eg. $\neg(X > a) = (X \leq a)$).

6 Conclusion and perspectives

What has to be retained from this paper is that a new generation of constraint solver – based not anymore on chronological backtracking but on the *Dynamic Backtracking* algorithm – is now emerging. Several seminal works contributed to the emergence of that new generation of constraint solvers and [Jussien, 1997] contributed to put those trends to work together.

The first experiments we performed were very promising. Of course, numerous other experiments on a large scale have to be done to ensure the reliability of the approach. Of course, many extensions of this work can be done. Our conviction is that the works presented in this paper open the door to a new and exciting sub-area in constraint solving. Time to be dynamic has arrived!

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.
- [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 also available as Technical Report 97-6-INFO at the 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*, Port Jefferson, N.Y., oct 1997. MIT Press.
- [Jussien and Boizumault, 1997c] Narendra Jussien and Patrice Boizumault. Stratégies en meilleur d’abord pour la relaxation de contraintes. In *Journées Francophones de Programmation en Logique et avec Contraintes*, Orléans, May 1997.
- [Jussien, 1997] Narendra Jussien. *Relaxation de Contraintes pour les problèmes dynamiques*. PhD thesis, Université de Rennes I, 1997. à paraître.
- [Lhomme *et al.*, 1996] O. Lhomme, A. Gotlieb, M. Rueher, and P. Taillibert. Boosting the interval narrowing algorithm. In *Joint International Conference and Symposium on Logic Programming*, pages 378–392, Bonn, September 1996. MIT Press.
- [Lhomme *et al.*, 1998] O. Lhomme, A. Gotlieb, and M. Rueher. Dynamic optimization of interval narrowing algorithms. *Journal of Logic Programming*, 1998. to appear.
- [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.