

Dynamic domain splitting for numeric CSPs

Narendra Jussien and ¹ Olivier Lhomme

Abstract. In this paper, a new search technique over numeric CSPs is presented: *dynamic domain splitting*. The usual search technique over numeric CSPs is a dichotomic search interleaved with a consistency filtering, which is called domain splitting. This paper proposes to replace chronological backtracking at the core of domain splitting by a non destructive backtracking technique.

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 may 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 respect to the domain reductions performed).

For example, [8, 16, 17] aim at defining concepts of higher order consistencies similar to *k*-consistency [6]. [4, 5] propose in a sense to merge the constraints about the same set of variables, giving one “total” constraint (thanks to numerical analysis techniques) and to perform arc-consistency on the total constraints. [1, 9] aim at expressing interval analysis prunings as partial consistencies.

Another direction of work over numeric CSPs is to try to optimize the computation of already existing consistency techniques. In this category, [19, 18] propose a method for dynamically detecting and simplifying cyclic phenomena during the running of a consistency technique, and [15] uses extrapolation methods for accelerating filtering techniques.

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 replaced by several candidates. For example:

- *intelligent backtracking* algorithms aim at backtracking in the search tree higher than chronological backtracking. An example is *conflict-directed backjumping* [21], which is able to identify the reasons of a failure and thus to backtrack directly to the relevant choice point.
- *dynamic backtracking* proposed in [7] aims at replacing backtracking (intelligent or not) by the suppression of the cause of the failure *without a complete 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 [21] and arc-consistency [22], *dynamic backtracking* has been improved with forward-checking [24], and [11, 12, 13] generalize *dynamic backtracking* plus arc-consistency for static CSPs and dynamic CSPs.

In this paper, we show how that idea of merging *dynamic backtracking* and arc-consistency can be useful for solving numeric CSPs, leading to an intelligent search tree exploration over the numeric CSPs.

The paper is organised as follows. Section 2 recalls the *dynamic backtracking* algorithm and summarizes some related works on the integration of constraint propagation in *dynamic backtracking*. Section 3 presents *dynamic domain splitting*, a new search-tree exploration for numeric CSPs. Finally, section 4 introduces first experimental results.

2 Background

2.1 Dynamic backtracking

Enumeration 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 [3]. A *nogood* is defined as a *globally inconsistent partial assignment* [25]. In other words, a partial assignment *A* is a *nogood* if and only if no solution contains *A*.

¹ École des Mines de Nantes – Département Informatique – 4 Rue Alfred Kastler – BP 20722 – F-44307 Nantes Cedex 3 – France

A first improvement of chronological backtracking has been provided through Prosser’s [21] *conflict-directed backjumping* (CBJ) algorithm. This algorithm implicitly uses nogoods by determining relevant backtrack points² without losing any solution.

Dynamic backtracking [7] goes further. Back jumping is interesting but a lot of possibly useful and independent 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 currently assigned to variable v in N . The key idea of *dynamic backtracking* relies in not performing any backtrack: variables chosen as *backtrack points* in CBJ are here unassigned without modifying any other assignment. That search processes by *jumps in the search space* rather than jumps in the search tree. 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*, and 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.

2.2 Dynamic backtracking + arc-consistency

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 have been proved not to appear in any solution of the considered problem.

For example, arc-consistency has been merged with the standard backtracking algorithm leading to the MAC algorithm (Maintaining Arc-Consistency) [23] 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.

Constraint propagation has also been integrated with other enumeration algorithms.

Conflict-directed backjumping [21] 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. Arc-consistency also has been introduced within *conflict-directed backjumping*, giving the MAC-CBJ algorithm [22].

As for *dynamic backtracking*, Schiex and Verfaillie have presented an integration of forward checking in *dynamic backtracking* for static [25] and dynamic CSPs [24]. Jussien and Boizumault [11, 12, 13] have proposed a general scheme to relax constraints in a dynamic context, and also describe an instance of the scheme, which is dedicated to the solving of

² Apart from the last assigned variable.

dynamic CSPs: the DECORUM system (Deduction-based Constraint Relaxation Management). The algorithm at the core of the DECORUM system is in some sense *dynamic backtracking* with arc-consistency for symbolic static or dynamic CSPs [14].

A similar algorithm is presented in the next section for solving numeric CSPs: it uses a dichotomic version of *dynamic backtracking* together with interval propagation.

3 Dynamic backtracking-like enumeration over numeric CSPs

3.1 Consistency techniques for numeric CSPs

The numeric CSPs fundamental idea was popularized by [2]: the projection of a numeric constraint over the axes of its variables is computed with interval arithmetic. For instance, the classical 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 [16, 17], 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.

In this paper, we will only consider 2B-consistency³, which is a kind of arc-consistency restricted to the bounds.

3.2 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 identified and deleted without complete loss of the work made between the cause and the failure. Unlike *dynamic backtracking*, the enumeration process is 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. For example, a variable x may have at the same time the four following splitting constraints: $x \geq 10, x \leq 20, x \geq 15, x \geq 17$.

Integration of 2B-consistency filtering within dynamic backtracking can be explained in three steps, addition of a splitting constraint, identification of the cause of a failure, removing of the splitting constraint that is responsible for the

³ 2B-consistency states a local property on a constraint and on the bounds of the domains of its variables. Roughly speaking, a constraint c is 2B-consistent if for any variable x in c the bounds a and b of the domain $D_x = [a, b]$ have a support in the domains of all other variables of c .

failure. They correspond respectively in the dynamic backtracking algorithm to the assignment of a value to a variable, the identification of the variable assignment that causes the failure and the unassignment of that variable.

3.3 Addition of a splitting constraint

As told above, an assignment in *dynamic backtracking* is replaced by the addition of a splitting constraint in the *dynamic domain splitting* algorithm. After each addition of a splitting constraint a propagation step is performed to enforce 2B-consistency.

3.4 Identification of the cause of a failure

Identifying the splitting constraint to reconsider upon a failure can be done in a way similar as MAC-CBJ identifies the point where to backtrack [22].

2B-consistency filtering algorithm works by shrinking the domains, which are intervals: it does not remove values one by one but it removes a subinterval either from the left of the interval or from the right. That is it removes in a single step a set of values.

To make 2B-consistency filtering algorithm capable of identifying precisely the cause of a failure, two explanations 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. Then, it suffices to modify the 2B-consistency filtering algorithm in such a way that each time a bound of a domain is changed (a set of values is removed from a domain), an explanation for the new bound is computed and kept.

More precisely, let $c(x_1, \dots, x_k)$ be a constraint over k variables, let b_i be one of the bounds of the variable x_i . Assume that, when computing the projection of c over x_i , a new value for b_i is obtained. Let B be the set of all the bounds of the domains of x_1, \dots, x_k that have been used in the computation of the new value for b_i , let E_b denote the explanation for the current value of the bound b , then the new explanation associated to b_i can be computed as $E_{b_i} = \bigcup_{b \in B} E_b$.

For example, let us consider the constraint $x = y + z$. Let $[x_L, x_U]$, $[y_L, y_U]$ and $[z_L, z_U]$ be the domains of x , y and z . A new upper bound for x can be computed if $x_U > y_U + z_U$: the new upper bound for x is $y_U + z_U$ and its explanation is $E_{y_U} \cup E_{z_U}$.

In MAC-CBJ, an explanation is a set of variables that have been assigned by the enumeration algorithm. From a variable, it is then easy to retrieve the assignment of this variable. In a dichotomic enumeration, a same variable can be split several times. So, it is not sufficient to keep only the variable. For that reason, an explanation in *dynamic domain splitting* will be a set of splitting constraints.

A failure occurs only when a domain becomes empty during the 2B-consistency filtering. An explanation for an empty domain, and hence for a failure, is simply the union of the explanations of the two bounds of this domain.

Now, we have an explanation of a failure but this is not exactly what we have called the *cause* of the failure. In MAC-CBJ as in *dynamic backtracking* the cause of the failure (the point to which to backtrack in MAC-CBJ and the variable to unassign in *dynamic backtracking*) is simply the last assigned

variable occurring in the explanation of the failure: it can easily be seen that backtracking upon a variable assigned after this one will not avoid the failure. Conversely, backtracking upon a variable assigned before this one may lose a solution. In the *dynamic domain splitting* algorithm, this is the same: the cause of the failure is the most recent splitting constraint in the explanation of the failure.

3.5 Removing the splitting constraint that is responsible for the failure

Dynamic backtracking is capable of unassigning a variable i while leaving untouched all the other variables thanks to eliminating explanations: all eliminating explanations containing i are deleted and the corresponding values are put back in their respective domains.

Dynamic domain splitting does not unassign a variable but removes a splitting constraint c . The difference is that the propagation step may have deeply propagated the effect of c . It is necessary to be able to undo only the effects of c without undoing the effects of the others splitting constraints *i.e.* the algorithm must be able to achieve 2B-consistency as if c had never existed and without a complete reexecution in order to keep good performances.

Remember that an explanation is kept for each current bound of the domains to allow the identification of the splitting constraint to remove upon a failure. Those explanations also make *dynamic domain splitting* capable of determining which set of values are to be put back into the considered domain. Let c be the splitting constraint to remove. All the bounds of the domains that have been reduced thanks to the splitting constraint c have c in their explanations. Thus, undoing the effects of c can be done simply by:

1. deleting all explanations containing c
2. restoring some previous values for the corresponding bounds.

Some values that have been put back in their domains could be possibly deleted by other splitting constraints. So, after having undone the effects of c a propagation step is needed to achieve 2B-consistency.

3.6 2B-consistency filtering with explanations

For the sake of clarity, the above three steps – namely, addition of a splitting constraint, explanation for a failure, removing a splitting constraint – can be presented through a 2B-consistency filtering algorithm called 2B-E (for 2B-consistency filtering with explanations).

2B-E has the following functionalities:

- Propagation of a new constraint in order to achieve 2B-consistency:
`addConstraint(in c, out fail-expl)`.
 That function modifies the domains as side effects. It returns true if there is no empty domain and false otherwise. In this latter case, it gives an explanation for the failure.
- “Depropagation” of a constraint:
`removeConstraint(in setOfCts, out fail-expl)`.
 The past effects of the constraints in `setOfCts` are deleted

and 2B-consistency is achieved. That function also modifies the domains as side effects, returns true if and only there is no empty domain and gives an explanation for a failure. A precondition for using this function is that all the constraints c in `setOfCts` must have been added by `addConstraint`.

2B-E maintains for each bound of the domains an explanation for its current value. The explanation is built as explained in subsection 3.4. An explanation is a set of splitting constraints. It is assumed in the rest of the paper that the initial CSP (the one considered by the first call to `addConstraint`) is 2B-consistent and that each bound of the domains has \emptyset as initial explanation. Thus, the explanation of a bound is the empty set if and only if the current value of the bound is due only to the constraints of the CSP and not to some splitting constraints.

3.7 Implementation sketch

Dynamic domain splitting works by splitting the domains in two. 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 (*resp.* c_2) is the negation of c_2 (*resp.* c_1), and we note $c_1 = \neg c_2$.

The main loop (*cf.* Figure 1) 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: it chooses the first part of the domain.

If a failure is detected, i.e. a domain becomes empty (lines 4, 5), the function `removeMostRecentSplit` is called with the failure explanation as parameter. If the failure explanation is an empty set, that means the constraint system has no solution (lines 10, 11). Else, the most recent splitting constraint c is removed with all its effects. All we have to do is, when removing the effects of a constraint c (lines 13 to 18), to remove all the “descendants” of the constraint, *i.e.* the splitting constraints on the same domain that are subsequent to c . Note that when removing a constraint (line 16), a failure may occur, in which case a recursive call to `removeMostRecentSplit` is needed (line 17).

Line 19 verifies that the constraints in $E \setminus \{c\}$ have not been removed, what could have been done by the recursive call of line 17 (if a constraint in $E \setminus \{c\}$ has been removed, it is not possible to keep any eliminating explanation for c , and the function terminates).

Then the negation of c (corresponding to the other part of the domain) either has an eliminating explanation or not:

$\neg c$ 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 computing the union of the eliminating explanations of the two splitting constraints c and $\neg c$, and by recursively calling the function `removeMostRecentSplit` with that parameter.

$\neg c$ has no eliminating explanation This means $\neg c$ has not been tried (in this context), then it has to be. First (line 23) the eliminating explanation of c , which is the failure explanation minus the constraint (see section 2.1), has to be kept. Then the negation splitting constraint $\neg c$ that corresponds to the other part of the domain is added.

```

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( $c$ , fail-expl) then
(5)       removeMostRecentSplit(fail-expl)
(6)     endif
(7)   endwhile
(8) end

procedure removeMostRecentSplit(in  $E$ )
(9) begin
(10)  if  $E = \emptyset$  then
(11)    no solution
(12)  else
(13)     $c \leftarrow$  the most recent splitting constraint in  $E$ 
(14)     $S \leftarrow \{c\} \cup$  all the splitting constraints descendant of  $c$ 
(15)    delete all elim. expl. containing an element of  $S$ 
(16)    if not removeConstraint( $S$ , fail-expl) then
(17)      removeMostRecentSplit(fail-expl)
(18)    endif
(19)    if  $E \setminus \{c\}$  is valid then
(20)      if  $\neg c$  has an eliminating explanation then
(21)        removeMostRecentSplit(expl( $\neg c$ ) $\cup$ ( $E \setminus \{c\}$ ))
(22)      else
(23)        keep  $E \setminus \{c\}$  as eliminating explanation for  $c$ 
(24)        if not addConstraint( $\neg c$ , fail-expl) then
(25)          removeMostRecentSplit(fail-expl)
(26)        endif
(27)      endif
(28)    endif
(29)  endif
(30) end

```

Figure 1. Dynamic Domain Splitting

Completeness and termination can be shown as completeness and termination of *dynamic backtracking*. A difficulty may come from the fact that domains are continuous. In fact, in practice we stop to add splitting constraints when all the domains are sufficiently tight. So we need to incorporate this assumption in order to derive the proof.

4 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.

Thus, the size of the search space generated by a union of disjoint subproblems is proportional to the sum of the search space generated by each subproblem independently. In [20], such a behavior, is called polynomial space aggressive.

Our first experiments have shown this behavior over several constraint systems, leading to drastic improvement in performance. For example, consider the following CSP:

$$\begin{aligned} x1 + x2 + x3 + x4 &= xl, y1 + y2 + y3 + y4 = yl \\ x1 + x2 - x3 + x4 &= 3, y1 + y2 - y3 + y4 = 3 \\ (x1^2) + (x2^2) + (x3^2) + (x4^2) &= 4 \\ (y1^2) + (y2^2) + (y3^2) + (y4^2) &= 4 \\ (x1^2) + (x2^2) + (x3^2) + (x4^2) - 2 * x1 &= 3 \\ (y1^2) + (y2^2) + (y3^2) + (y4^2) - 2 * y1 &= 3 \\ x1, x2, x3, x4, y1, y2, y3, y4 &\in [-10^{10}, +10^{10}] \\ xl, yl &\in [-0.2, 0.2] \end{aligned}$$

This is an artificial constraint system composed of twice the same constraint system S_x and S_y , without any shared variable. Moreover S_x and S_y are underconstrained. We try to reach only one solution with a precision of 10^{-5} . The commercial product Numerica [10] has been run more than 300 hours on an UltraSparc without giving a solution. With *dynamic domain splitting* a solution can be obtained in a few seconds.

Of course, in this case a preprocessing could decompose the constraint system in S_x and S_y , leading to an independent solving of S_x and S_y , and the problem would be solved easily even with a backtracking based search. But in general, the independence of subsystems is more subtle, and can be observed only dynamically.

Nevertheless, *dynamic domain splitting* is not the panacea. When thrashing as above does not occur, there may exist a substantial overhead in time. We have tried *dynamic domain splitting* over a few traditional benchmarks, which typically are strongly connected. The overhead may vary from a factor of 1 to 600 compared with Numerica. Experiments need to be done on a more large scale, with an optimized implementation.

5 Conclusion and perspectives

This paper has presented a new enumeration algorithm for solving numeric 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. This has already been done for symbolic dynamic CSPs in the DECORUM system which handles over-constrained problems arising in dynamic environments [11, 12, 13].

REFERENCES

[1] F. Benhamou, D. Mc Allester, and P. Van Hentenryck, 'CLP(Intervals revisited)', in *International Logic Programming Symposium*. MIT Press, (1994).
 [2] E. Davis, 'Constraint propagation with interval labels', *Artificial Intelligence*, **32**(2), 281–331, (1987).

[3] J. Doyle, 'A truth maintenance system', *Artificial Intelligence*, **12**, 231–272, (1979).
 [4] B. Faltings, 'Arc consistency for continuous variables', *Artificial Intelligence*, **65**(2), (1994).
 [5] B. Faltings and Esther Gelle, 'Local consistency for ternary numeric constraints', in *IJCAI*, (August 1997).
 [6] Eugene Freuder, 'Synthesizing constraint expressions', *Communications of the ACM*, **21**, 958–966, (November 1978).
 [7] Matthew L. Ginsberg, 'Dynamic backtracking', *Journal of Artificial Intelligence Research*, **1**, 25–46, (1993).
 [8] D. Haroud and B. Faltings, 'Consistency techniques for continuous constraints', *Constraints*, 1(1–2):85–118, (1996).
 [9] P. Van Hentenryck, D. Mc Allester, and D. Kapur, 'Solving polynomial systems using branch and prune approach', *SIAM Journal*, **34**(2), (1997).
 [10] P. Van Hentenryck, P. Michel, and L. Deville, *Numerica, a modeling language for global optimization*, MIT press, 1997.
 [11] Narendra Jussien, *Relaxation de Contraintes pour les problèmes dynamiques*, Ph.D. dissertation, Université de Rennes I, 1997. in french.
 [12] 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).
 [13] 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.
 [14] 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).
 [15] Y. Lebbah and O. Lhomme, 'Acceleration methods for numeric csp's', in *Proceedings of AAAI-1998*, (1998).
 [16] O. Lhomme, 'Consistency techniques for numeric CSPs', in *IJCAI'93*, pp. 232–238, Chambéry, France, (August 1993).
 [17] O. Lhomme, *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles*, Ph.D. dissertation, Université de Nice – Sophia Antipolis, 1994.
 [18] O. Lhomme, A. Gotlieb, and M. Rueher, 'Dynamic optimization of interval narrowing algorithms', *Journal of Logic Programming*, Forthcoming, (1998).
 [19] O. Lhomme, A. Gotlieb, M. Rueher, and P. Taillibert, 'Boosting the interval narrowing algorithm', in *Proc. of the 1996 Joint International Conference and Symposium on Logic Programming*, pp. 378–392. MIT Press, (1996).
 [20] David A. McAllester, 'Partial order backtracking', Research report, ft.ai.mit.edu:/pub/dam/dynamic.ps, (1993).
 [21] 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).
 [22] Patrick Prosser, 'MAC-CBJ: maintaining arc-consistency with conflict-directed backjumping', Research Report 95/177, Department of Computer Science – University of Strathclyde, (1995).
 [23] Daniel Sabin and Eugene Freuder, 'Contradicting conventional wisdom in constraint satisfaction', in *Principles and Practice of Constraint Programming*, ed., Alan Borning, volume 874 of *Lecture Notes in Computer Science*. Springer, (May 1994). (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).
 [24] Gérard Verfaillie and Thomas Schiex, 'Dynamic backtracking for dynamic csp's', in *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*, eds., Thomas Schiex and Christian Bessière, Amsterdam, (August 1994).
 [25] 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).