

Dynamic Backtracking with Constraint Propagation

Application to static and dynamic CSPs

Narendra Jussien and Patrice Boizumault
École des Mines de Nantes – Computer Science Department
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 behaviour that consists in local modifications of the choices made up to the current situation.

1 Introduction

Many industrial and engineering problems can be considered as constraint satisfaction problems (CSPs). A CSP is defined by a set of variables each with an associated finite domain of possible values and a set of constraints on the variables.

Such problems are solved using enumeration techniques based mainly upon standard backtracking. Standard backtracking presents several drawbacks that have been handled through various improvements leading to Intelligent Backtracking techniques (*eg.* Conflict-directed BackJumping [Prosser, 1993] and Nogood Recording [Schiex and Verfaillie, 1994]). [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 the one hand, [Sabin and Freuder, 1994] showed that constraint propagation (arc consistency) was well worth integrating with standard backtracking. On the other hand, [Bessière and Régin, 1996] showed that constraint propagation was not well worth integrating within conflict-directed backjumping [Prosser, 1995] cutting down interests in this field. What about integrating constraint propagation with dynamic backtracking? 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].

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 Arc Consistency within Dynamic Backtracking.

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 static CSPs. Section 5 gives experimental results comparing our method with MAC for solving static problems. For structured problems, our proposition has a similar behaviour

or is better than the MAC algorithm. Section 6 shows the extension to dynamic CSPs.

2 The DECORUM system

The DECORUM system handles 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 contain the constraint to relax and to repropagate the constraints upon the corresponding variables. The basic functionalities of an *ace* algorithm are then twofolds:

- propagation of a new constraint in order to achieve arc-consistency: function `addConstraint(c, E)`, with as parameters an explanation E for the addition of the constraint c . When a contradiction occurs during the computation of arc-consistency, the *ace* algorithm gives an explanation for it.
- “depropagation” of a constraint: function `removeConstraint`. The past effects of the constraints are deleted and arc-consistency is achieved.

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

¹It must be partly responsible for the contradiction.

²Some constraints are more important than others.

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 to 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 lead 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 v_i in $Values$ (11) assign v_i 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 v_i in $Values$ (12) assign v_i 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 avoids the exploration of useless alternatives in the search tree but does not avoid *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.

4 Dynamic Backtracking + Arc Consistency

Constraint propagation for static CSPs 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

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)       $Values \leftarrow D_i \setminus e(i)$ 
(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  $v_i$  in  $Values$ 
(17)       assign  $v_i$  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

idea is to propagate constraints (*i.e.* to achieve arc-consistency) after each instantiation.

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]. 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 have never been made and without a complete reexecution in order to keep good performances.

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. [Verfaillie and Schiex, 1994] have observed that forward checking was apparently much more beneficial to Intelligent Backtracking algorithms than to dynamic backtracking. 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

```

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 of  $NG \setminus \{j\}$ 
(15)      endif
(16)    else
(17)      select  $v_i$  in  $Values$ 
(18)      if adding constraint  $i = v_i$  is contradictory then
(19)        set  $i \neq v_i$  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( $i = v_i, i$ ) 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)      $S \leftarrow$  all the enumeration constraints whose
(18)     explanation contains  $j$ 
(19)     if not removeConstraint( $S \cup \{j = v_j\}$ ) then
(20)        $E' \leftarrow$  contradiction explanation
(21)       unassignMostRecent( $E'$ )
(22)     endif
(23)     if  $E \setminus \{j\}$  is valid then
(24)       if not addConstraint( $j \neq v_j, E \setminus \{j\}$ ) then
(25)          $E' \leftarrow$  contradiction explanation
(26)         unassignMostRecent( $E'$ )
(27)       endif
(28)     endif
(29)   endif
(30) end

```

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

on the behaviour 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 another 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 behaviour can possibly lead to a contradiction. Thus, the propagation step must then be achieved until consistency is reached (relaxing constraints as is done between lines 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 Experiments on static CSPs

In this section, we compare DECORUM with the MAC algorithm. In order to make accurate comparisons independent from implementations, we implemented MAC and DECORUM using the same data structures and overall algorithms. These implementations are not state-of-the-art versions of the considered methods. They provide a convenient way of comparing brute performance on problems without any implementation bias.

We studied these two algorithms on random problems. Classically random CSPs are generated considering four parameters: the number n of variables, the uniform size d of their domains, the density p_1 of the problem and the tightness p_2 of the constraints of the problem. We used the random problems generator given by Bessière, Dechter, Frost and Régin (*cf.* [Bessière and Régin, 1996]).

We ran two related sets of experiments. The first set has been driven on random problems. The second set involves structured problems that take advantage of the specificity of the search provided by DECORUM.

5.1 Comparing results on random problems

We present here results obtained for random $(15, 10, 0.48, p_2)$ problems where p_2 varies from 0.4 to 0.6. The expected phase transition⁴ is for $p_2 = 0.49$. For each value of p_2 , the average number of unassignments⁵, constraint checks and **cpu** time were recorded for 35 instances⁶ of such problems.

For each series of problems, four different dynamic variable orderings are considered : the smallest available variable index (**no**), the smallest domain size (**dom**), the smallest domain size vs degree of the variable ratio (**dom/deg**) and the greatest degree for a variable (**deg**). Bessière and Régin [1996] have shown that the best one for the MAC algorithm was **dom/deg** for random problems.

Figure 4 shows the results obtained for the MAC algorithm⁷ and figure 5 shows the results for our approach (DECORUM).

These results clearly show the same overall behaviour in terms of constraint checks, backtracks and **cpu** time for the two methods whatever. The **no** variable ordering is inefficient for both approaches. From these first results, no distinction can be made between the three other orderings.

Figure 6 shows a comparison between the MAC algorithm and the DECORUM approach for solving the same problems and using the **dom/deg** variable ordering (curves are similar for the other orderings).

Let us first consider the number of backtracks. We can see in figure 6 that our approach does not improve the number of backtracks for the search. This is mainly due to the random nature of the treated problems. Obviously, no interesting information can be deduced from the search. Such a bad result necessarily leads to more constraint checks because a constraint removal is more costly than a mere *backtrack* (due to the computation of arc-consistency filtering). This is confirmed on the constraint checks graph of figure 6.

So, when random problems are solved, few information can be deduced from the search. In that situation, dynamic backtracking with constraint propagation

⁴A peak in the activity (number of constraint checks) is identified for specific values of the tightness of the constraints when n , d and p_1 are held constant [Prosser, 1994].

⁵In the following, we will call that number the number of backtracks as it is done in the CLP community.

⁶Considering more than 30 random instances of a given problem allows the use of statistical tests to compare results.

⁷As shown in this figure, the three results are strongly related when using the MAC algorithm. From now on, when reporting MAC results, we will only present results for the number of backtracks which will reflect the behaviour for the other results.

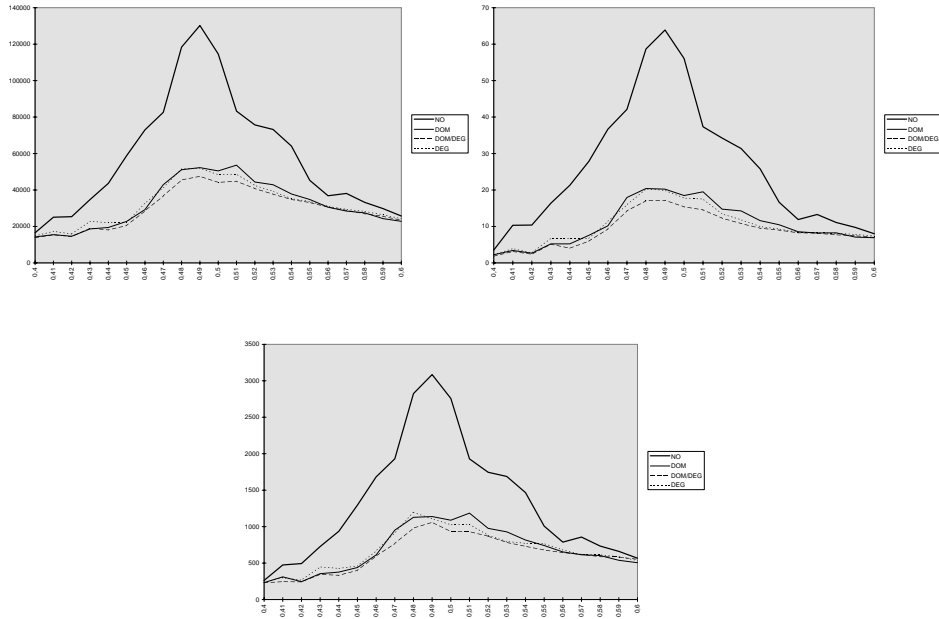


Figure 4: MAC – Results for $\langle 15, 10, 0.48, p_2 \rangle$ problems (*resp.* constraint checks, backtracks and **cpu** time)

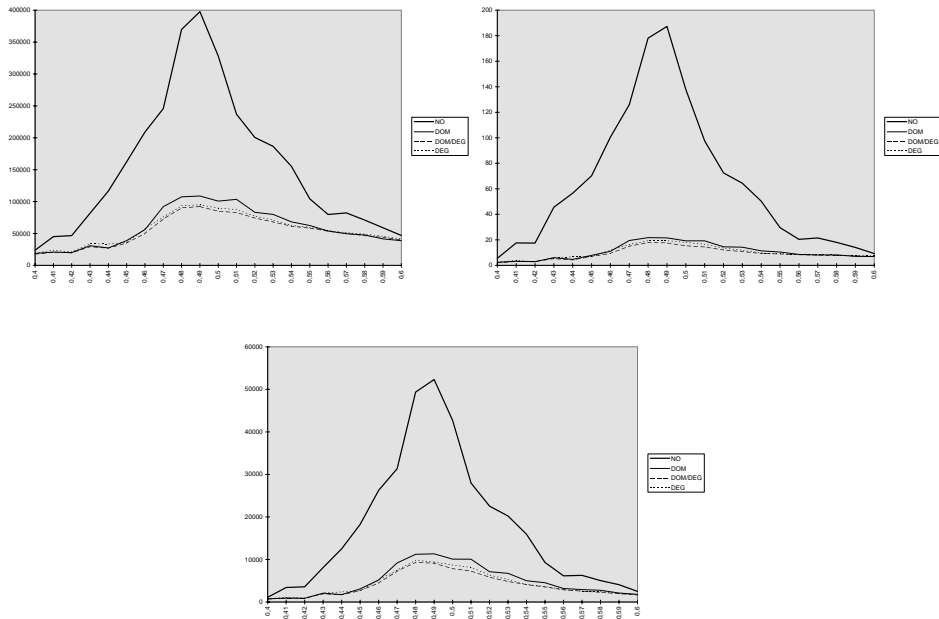


Figure 5: DECORUM – Results for $\langle 15, 10, 0.48, p_2 \rangle$ problems (*resp.* constraint checks, backtracks and **cpu** time)

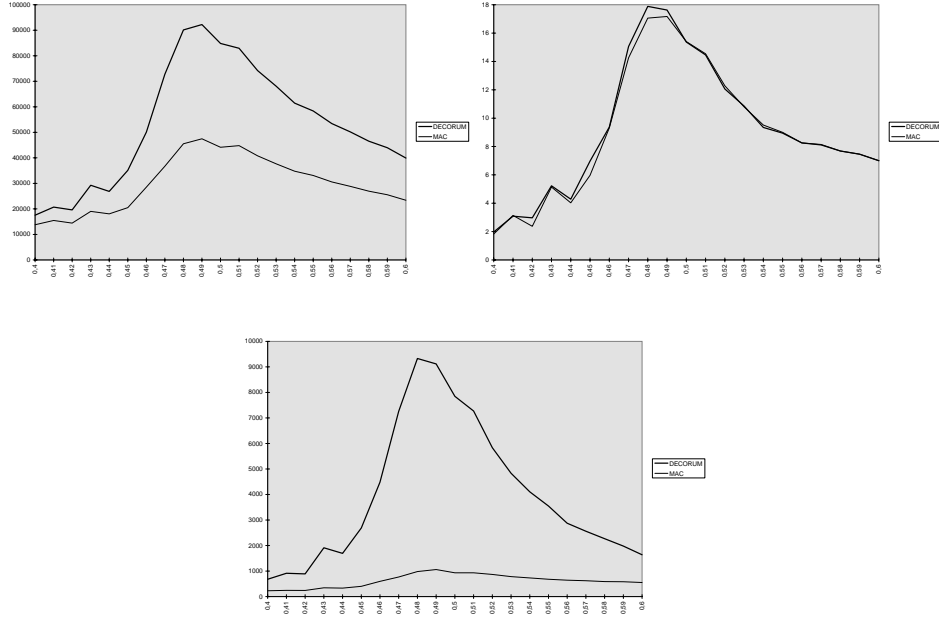


Figure 6: Comparison DECORUM- MAC - Results for $(15, 10, 0.48, p_2)$ problems (*resp.* constraint checks, backtracks and **cpu** time) with the **dom/deg** variable ordering

presents an overhead in constraint checks and **cpu** time.

5.2 Solving structured problems

We are currently working on real life dynamic resource allocation problems. The characteristics of these problems give them a *structure*: many poorly related sub-problems are solved altogether. Constraints and variables are clustered in quasi-independent sets.

Dynamic backtracking + constraint propagation is well suited for handling structured problems from which relevant backtracking information can be extracted.

From the random problems presented in the previous section, we build two kinds of *structured* problems.

- in the first series, we merely *duplicate* twice the problem *i.e.* we define the problem to solve as thrice the random problem obtained in the previous section. This leads to solve a structured problem presenting three independent self-connected sets of variables. In order to prevent any identification technique of independent sets of variables we arbitrarily post a *fake* constraint between any of the variables of the three components.

Variables are shuffled *i.e.* variable v_1 corresponds to the first variable of the first instance of the original problem, variable v_2 corresponds to the first variable of the second instance of the original problem and so on.

The resulting series called “*duplicate problems*” in the following thus involves $(30, 10, 0.23, p_2)$ structured problems. Solving such a problem with a standard backtracking-based technique without identifying the independent components may lead to an explosion since work on one problem may be *destroyed* by backtracking on variables of another problem.

- in the second series, we introduce an over-constrained sub-problem within the problem definition. That over-constrained problem consists in 5 variables with a same domain of size 4 upon which a two-by-two difference is posted ⁸. That sub-problem is added at the end of the definition of the original problem (one of the problems generated in the previous section).

These problems are called “*contradiction included problems*” in the following. Solving such a problem with a standard backtracking-based technique without identifying the over-constrained subproblem will lead to an explosion since all possible solutions for the random problem have to be determined in order to prove the inconsistency of the whole problem.

These experiments are not meant to be realistic *per se* but are designed to show the benefits of dynamic backtracking + arc consistency. If standard backtracking is used to solve the first series and if the variables are not enumerated set by set, there will be an explosion of the number of required backtracks because independent work on one set will be undone when backtracking on another set of variables. As for the second series, if the variables of the contradictory sub-problem are enumerated after variables from the first problem, a standard backtracking-based search will lead to the exploration of all the possible solutions of the random problem even if the two problems are completely disconnected. Non backtrack-based search techniques can be of great help for such structured problems.

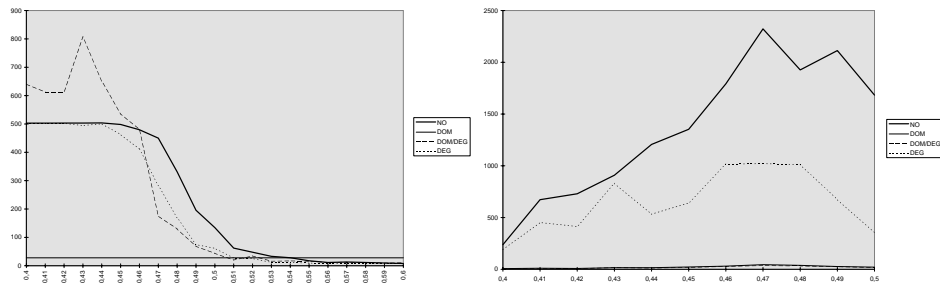


Figure 7: MAC – Number of backtracks for contradiction included problems (left) and duplicate problems (right)

Figure 7 shows the results obtained with MAC for the two series. In the presented results, computations have been resumed after 2500 backtracks for the *duplicate* problems (the limit was at 800 backtracks for *contradiction included* problems). Therefore, these results are only a lower bound of the real results for problems where p_2 is in $[0.4, 0.50]$ for all the results using MAC *i.e.* for each ordering and for value of p_2 in that interval, there is at least one problem whose resolution exceeds the authorized limit.

Regarding to the contradiction included problems, the decreasing behaviour of the ordering methods is due to the fact that as p_2 increases, the number of consistent *real* problems falls as soon as the original is shown to be inconsistent; the search ceases even if the over-constrained sub-problem has not been tackled. Therefore, when comparing with DECORUM we will focus our study for values of p_2 lower than 0.5.

Figure 8 shows the results obtained by DECORUM on the same problems.

⁸A efficient way to solve it would be to use a global cardinality constraint [Régis, 1994], [Régis, 1996]. but our goal is just to introduce an inconsistent sub-problem that cannot be easily determined.

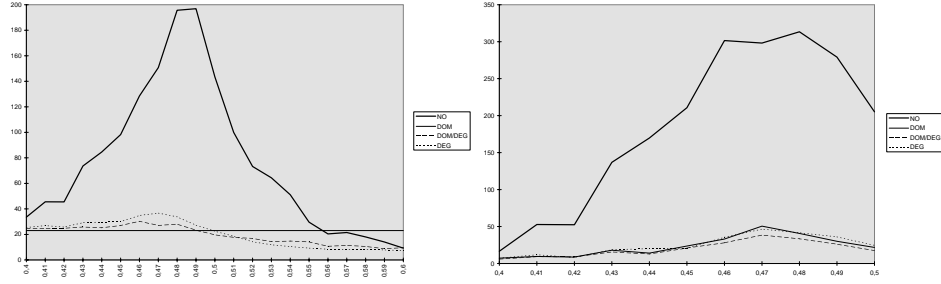


Figure 8: DECORUM – Number of backtracks for contradiction included problems (left) and duplicate problems (right)

For the two series, the results are similar to those observed for the original problems (see figure 5). The overall behaviour is the same. A close look comparing figures 5 and 8 shows that the results for the “duplicate problems” are not more than tripled with DECORUM and results for “contradiction included problem” are translated towards the top of the graph. Thanks to the explanations, DECORUM takes dynamically advantage of the structure of the problem.

Figure 9 compares results between MAC and DECORUM on duplicate problems when using the `deg` variable ordering. For convenience, we also report the results obtained by the two methods when solving the original problems.

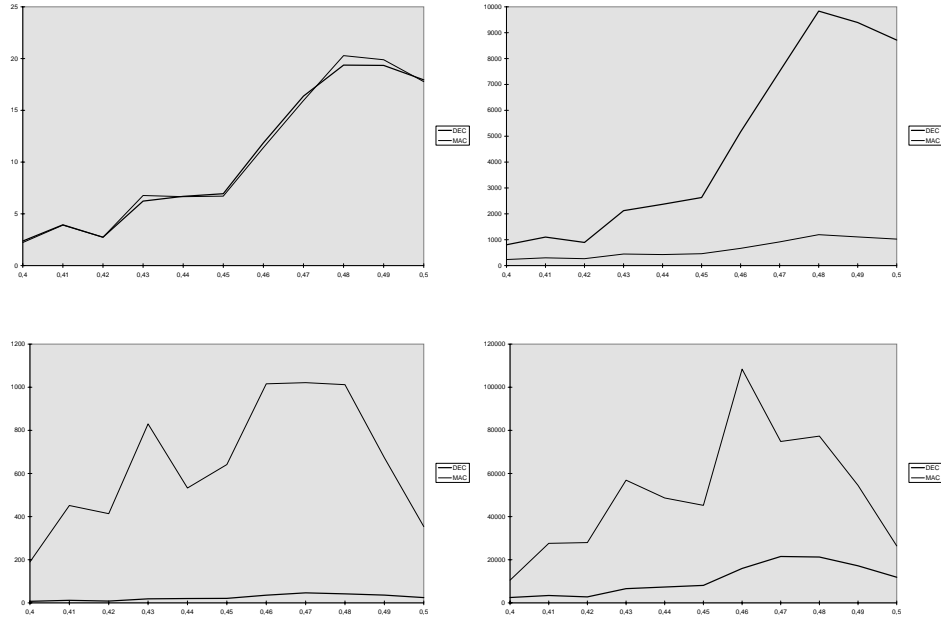


Figure 9: Comparison DECORUM vs MAC – Results (*resp.* backtracks and `cpu` time) for original problems (first row) and duplicate problems (second row) with the `deg` variable ordering

When the problem to be solved has a structure, the overhead induced by the use of DECORUM is overridden by the benefits extracted from the *intelligence* of the performed search.

Furthermore the reported results for MAC are only lower bounds for the values of interest for p_2 . When considering other variable orderings, MAC and DECORUM give similar results in number of backtracks.

Let us now compare results for the second series of structured problems. Figure 10 presents the results obtained when comparing MAC and DECORUM when using the **dom/deg** ordering on the contradiction included problems.

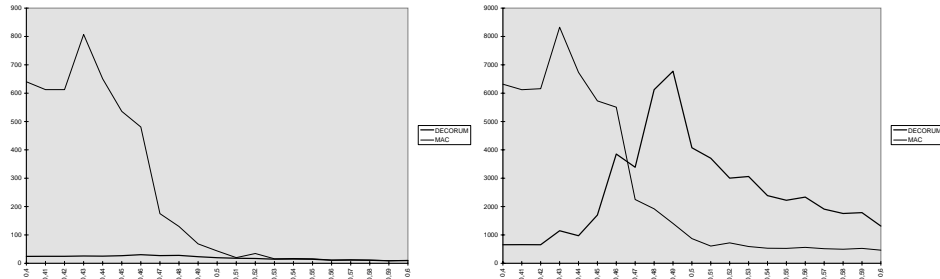


Figure 10: Comparison DECORUM vs MAC – Results (*resp.* backtracks and **cpu** time) for contradiction included problems with the **dom/deg** variable ordering

DECORUM provides comparable and even better results than MAC on this class of problem depending on the variable ordering. The inherent overhead of DECORUM is compensated by the efficiency of the provided search. The crossing of the two curves in figure 10 is explained by the fact that when approaching the critical tightness (here $p_2 = 0.49$) more original problems become over-constrained and thus the added over-constrained sub-problem is not even considered by the solver before halting the computation due to a contradiction.

5.3 First conclusions

These experimental results show that integrating constraint propagation with Dynamic Backtracking can greatly improve the search compared to the MAC algorithm despite the necessary overhead induced by the management of the explanations. These experiments show that it is possible to provide an efficient integration of constraint propagation within the Dynamic Backtracking scheme. It would be interesting to study when exactly the DECORUM search performance overcomes the necessary overhead.

6 Handling dynamic CSPs

The extension to dynamic CSPs is based on the fact that maintaining explanations for the domain reductions aims to make dynamic any static algorithm; roughly speaking, it is sufficient to forget reductions whose explanations contains the constraint to relax and to repropagate the constraints upon the corresponding variables.

6.1 Explanations

The *eliminating explanations* of dynamic backtracking are extended in order to maintain arc-consistency (particularly to perform constraint deletion).

- a deduction $\delta_{i \neq v_i}$ is the removal of the value v_i from the domain of variable i .

- An explanation for a deduction $Expl(\delta_{i \neq v_i})$ is a couple $(C, \{\delta_{X_i \neq v_i}\})$ such that :
 $(j \neq v_j) \wedge \dots \wedge (k \neq v_k) \wedge C \rightarrow (i \neq v_i)$
- like dynamic backtracking [Ginsberg, 1993], only one explanation is kept for a deduction.

This notion of explanation is similar to those of Schiex and Verfaillie for dynamic backtracking + forward checking for dynamic CSPs [Verfaillie and Schiex, 1994]. DECORUM will also use these explanations in order to maintain Arc-Consistency and to automatically determine the constraint(s) to relax.

6.2 Providing explanations

Explanations are computed by the arc-consistency algorithm. For example, let us consider an AC4-based algorithm. Such an algorithm proceeds in two phases when considering the addition of a constraint:

- In the first step, value removals directly due to the addition of the constraint are computed. The explanation is then the union of the removal of the supported values of the considered one plus the new constraint.
- In the second step, value removals are propagated. The explanation here is the union of the removal of the supported values of the newly considered removal.

Consider a CSP with n variables whose domains are of maximum size d , and e constraints. As only one explanation at a time is kept for any deduction, the space complexity is $O(n \times d \times (n + e))$: the maximum size of an explanation is $O(e + n)$ and the number of valid value removals is in the worst-case: $O(n \times d - n + 1)$ ⁹. Hence, the number of explanations is in the worst-case $O(n \times d - n + 1)$.

6.3 Removing a constraint

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 removals have to be reconsidered.

Operationally, the associated values of the set $\alpha(c)$ are put back in their respective domains¹⁰. 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.

6.4 Modeling the search

The main idea in integrating constraint propagation within Dynamic backtracking is to consider assignment and unassignment as constraint addition and removal. In other words, enumeration is considered as a dynamic process.

We consider the domain declaration of a variable through a new constraint: the **one-of** constraint. That constraint ensures that exactly one constraint of a given set is active at any moment of the computation. For example, when considering variable v with domain $\{1, 2, 3\}$ a constraint **one-of** $(v = 1, v = 2, v = 3)$ is posted.

When enumerating, **one-of** constraints are activated selecting a value to be assigned to the associated variable¹¹. After each *assignment* a propagation step

⁹only one remaining value in each domain and one empty domain.

¹⁰In 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

¹¹At this step, any value ordering technique can be used.

is considered through an ACE algorithm (an equality constraint has been added) providing *explanations* throughout the computation.

If a contradiction occurs, a contradiction explanation is computed. In that explanation, we select the most recent assignment constraint involved. If none exists the problem is globally inconsistent. Else, the associated variable is unassigned (the associated constraint is removed using the ACE algorithm) and another value for that variable is tested, while the other variables remain assigned.

The completeness of our approach proceeds from the completeness of Dynamic Backtracking. In order to ensure the completeness of the approach, we record removal explanations for all removed constraint. These are our “*eliminating explanations*”. Let us consider a contradiction explanation E_c and a constraint $c \in E_c$ to be removed. A removal explanation for the removal of constraint c is then $E_c \setminus \{c\}$. This is very closed to the management of eliminating explanations.

7 Conclusion

In this paper, we presented the integration of constraint propagation within dynamic backtracking providing the DECORUM system. DECORUM have proved to be useful on *structured* problems as shown by the experimental results. A well mastered deduction maintenance system could be of great use for developing efficient search techniques.

Our further works include:

- solving real dynamic problems using DECORUM. Several projects are currently studied from the DECORUM point of view. They all involve dynamic resource allocations either for military purposes, for communication networks, ...
- using our method for solving over-constrained problem. Indeed, contradiction explanations can be used to determine constraints to relax when a contradictory problem is identified. That is one original purpose of the DECORUM system.

References

- [Bessière and Régim, 1996] Christian Bessière and Jean-Charles Régim. 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.
- [Debruyne, 1995] Romuald Debruyne. Les algorithmes d'arc-consistance dans les CSP dynamiques. *Revue d'Intelligence Artificielle*, 9(3), 1995.
- [Ginsberg, 1993] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [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. Stratégies en meilleur d'abord pour la relaxation de contraintes. In *Journées Franco-phones 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, 24 October 1997.
- [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, 1994] Patrick Prosser. Binary constraint satisfaction problems: Some are harder than others. In *ECAI'94*, pages 95–99, 1994.
- [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.
- [Régis, 1994] Jean-Charles Régis. A filtering algorithm for constraints of difference in CSPs. In *AAAI 94, Twelfth National Conference on Artificial Intelligence*, pages 362–367, Seattle, Washington, 1994.
- [Régis, 1996] Jean-Charles Régis. Generalized arc-consistency for global cardinality constraint. In *AAAI'96*, 1996.
- [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).
- [Schiex and Verfaillie, 1994] Thomas Schiex and Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2):187–207, 1994.
- [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 csp. 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.