Computing and Informatics, Vol. 40, 2021, 1056–1079, doi: 10.31577/cai_2021_5_1056

LAZY REPAIRING BACKTRACKING FOR DYNAMIC CONSTRAINT SATISFACTION PROBLEMS

Yosra Acodad, Amine Benamrane, El Houssine Bouyakhf

LIMIARF, Faculty of Sciences
Mohammed V University in Rabat, Morocco
e-mail: {yosra.acodad, benamraneamine}@gmail.com, bouyakhf@mtds.com

Imade Benelallam

INSEA, National Institute of Statistics and Applied Economic
Irfane Rabat, Morocco
e-mail: i.benelallam@insea.ac.ma

Abstract. Extended Partial Dynamic Backtracking (EPDB) is a repair algorithm based on PDB. It deals with Dynamic CSPs based on ordering heuristics and retroactive data structures, safety conditions, and nogoods which are saved during the search process. In this paper, we show that the drawback of both EPDB and PDB is the exhaustive verification of orders, saved in safety conditions and nogoods, between variables. This verification affects remarkably search time, especially since orders are often indirectly deduced. Therefore, we propose a new approach for dynamically changing environments, the Lazy Repairing Backtracking (LRB), which is a fast version of EPDB insofar as it deduces orders directly through the used ordering heuristic. We evaluate LRB on various kinds of problems, and compare it, on the one hand, with EPDB to show its effectiveness compared to this approach, and, on the other hand, with MAC-2001 in order to conclude, from what perturbation rate resolving a DCSP with an efficient approach can be more advantageous than repair.

Keywords: DCSP, LRB, extended PDB, pdeg heuristic, MAC-2001

Mathematics Subject Classification 2010: 97P50, 97R40

1 INTRODUCTION

In Artificial Intelligence (AI) and operation research, a large number of problems can be modeled as Constraint Satisfaction Problem (CSP). In fact, constraints are naturally present in real problems such as allocation problems [3], scheduling problems, sensor networks problems [18], etc. The concept of CSPs indicates all these problems and consists of looking for the solutions satisfying them. To increase the efficiency of the constraint satisfaction algorithms, many techniques as filtering, improved backtracking, using efficient representations and heuristics have been developed. However, these techniques assume that the complete description of the CSP instance is known and fixed in advance. This is a strong limitation when dealing with real situations [17], where problems may evolve due to changes in the environment or their execution conditions.

The notion of dynamic CSP (DCSP) [9] has been introduced to represent such situations. A DCSP is an extension of a static CSP. It can be viewed as a sequence of CSPs, where each one differs from the previous one by addition or removal of some constraints. In fact, all other possible changes of a CSP (constraints or domain modifications, variables additions or removals) can be expressed in terms of constraints additions or removals. The present paper focuses on scenarios where after solving a DCSP, some constraints have been added or changed, then, the old solution and reasoning are required to be effectively repaired.

Most DCSPs complete algorithms are based on backtracking, constraint propagation and nogood learning. The nogoods are used by intelligent back-trackers to record information regarding dead-ends in order to avoid encountering them again. They are used in Partial-order Dynamic Backtracking (PDB) [11], which considers a complete set of assignments that are incrementally modified until all constraints are satisfied, or no solution is found. The advantage of PDB is that it allows great flexibility in the backtracking strategy which affects the flexibility of the search space exploration. Extended Partial-order Dynamic Backtracking EPDB [2] exploits this flexibility by repairing solutions using variables ordering heuristics (VOHs) to build an optimal order, and by fixing the checked constraint, contrary to PDB, till its consistency.

In this paper, we propose a fast version of EPDB, the Lazy Repairing Backtracking (LRB) approach. This algorithm aims to repair solutions using EPDB advantages while eliminating unnecessary tests of orders.

This paper is organized as follows: First, we recall DCSP concept, EPDB approach and profound degree (pdeg) [1] heuristic. Then, we present the proposed LRB algorithm. Next, we evaluate the performance of this approach, compared to EPDB and MAC-2001 [4, 5] on different types of problems respectively in order to show its effectiveness and to demonstrate when repairing a DCSP is more efficient than resolving it from the scratch. Finally, we extract some conclusions and directions for further research.

This paper is organized as follows: In Section 2 we summarize some backround concept useful for the rest of the paper. Then in Section 3 we present and detail our

new repairing approch LRB. Next in Section 4 we present some empirical experiments to defend the performance of LRB approach. Before conclude, we discuss in Section 5 experiments' results in both repairing and solving configuration.

2 BACKGROUND

2.1 Dynamic Constraint Satisfaction Problem

Many real problems are dynamic, due to the environment or problem scenario changes. These changes are caused by the dynamism of the environment nature, spurious actions, incomplete knowledge, etc. A DCSP is a CSP which is adapted to changes. A problem change can be a restriction, which is an addition of constraints, or relaxation, that is an elimination of some existing constraints.

Definition 1 (CSP)**.** A Constraint Satisfaction Problem (CSP) is a tuple (X, D, C), where:

- X is a set containing n variables $\{x_1, x_2, \ldots, x_n\}$.
- D is a set of domains $\{D(x_1), D(x_2), \ldots, D(x_n)\}$ for these variables, such as each $D(x_i)$ contains the possible values which x_i may take.
- C is a set of m constraints $\{c_1, c_2, \ldots, c_m\}$ between variables in subsets of X. Each $c_i \in C$ expresses a relation defining which combinations of variables assignments are allowed for the variables $vars(c_i)$ in the scope of the constraint.

Definition 2 (DCSP)**.** Consider P a dynamic constraint satisfaction problem (DCSP) [9]. P is a sequence of static CSPs $P_0, \ldots, P_\alpha, P_{\alpha+1}, \ldots$, where each P_i differs from the previous P_{i-1} by the addition or removal of some constraints (we assume that all CSPs possible changes can be expressed in terms of constraint additions or removals).

If $P_\alpha = (X, D, C_\alpha)$, then $P_{\alpha+1} = (X, D, C_{\alpha+1})$, where $C_{\alpha+1} = C_\alpha \pm C$, such as C is a set of constraints.

2.2 Extended Partial-Order Dynamic Backtracking

The dynamic scheduling offers the possibility to rebuild a new order among variables during the search. The Partial-order Dynamic Backtracking (PDB) [11] is one of the approaches using this technique. However, it suffers from a lack of flexibility in backtracking strategy although using a partial order which supports dynamism.

In fact, PDB replaces the fixed order of variables, used in several approaches to solve CSPs, with a partial order that is dynamic and partially modified during the search process. It uses the concept of nogoods introduced in the Dynamic Backtracking algorithm (DBT) [10].

Definition 3 (Nogood)**.** A nogood [11] is an expression of the form:

$$(X_1 = v_1) \wedge \cdots \wedge (X_k = v_k) \Rightarrow X \neq v. \tag{1}$$

It is a failure explanation which can be used to represent a constraint as an implication. It is logically equivalent to the constraint:

$$\neg[(X_1 = v_1) \wedge \cdots \wedge (X_k = v_k) \wedge (X = v)] \tag{2}$$

that represents a set of assignments that cannot be extended to a solution of the problem.

The order in PDB is manipulated when a nogood is generated, in order to justify an elimination when a constraint is violated, or when a failure of instantiating a variable is met due to a domain wipe out. Contrary to DBT, when a nogood is induced during the search, PDB verifies the current order between variables concerned with this conflict. If no order is imposed, there may be significant choices for the variable that will represent the nogood conclusion. The approach performance depends mainly on the quality of this choice, which is not covered in PDB.

Generally, for two given variables, the order is not defined at the beginning. There is a partial order that is built during the search process, to achieve the solution if it exists, or to detect the unsatisfiability through the empty nogood.

The partial order between variables is imposed by saved nogoods and safety conditions, a data structure of orders presented by the deleted nogoods.

Definition 4 (Safety condition)**.** A safety condition [11] is defined as an assertion of the form: $x < y$, with x and y variables.

If S is a set of safety conditions, we denote by $\leq_s$ the transitive closure of $<$, meaning that S is acyclic if $\leq_s$ is anti-symmetric:

$$x <_s y \quad \Rightarrow \quad x \leq_s y \quad \& \quad y \not\leq_s x. \tag{3}$$

In another way, $x <_s y$ if there is a sequence (possibly empty) of safety conditions:

$$x < z_1 < z_2 < \cdots < z_n < y. \tag{4}$$

The first improvement of Extended Partial-order Dynamic Backtracking (EPDB) [2] is to extend the classical description of PDB, by exploiting its flexibility to repair assignments using ordering heuristics to control changes. Thus, in a conflict situation, when no order exists between variables in conflict, EPDB changes the less relevant sub-problem. Otherwise, it keeps as long as possible the most difficult sub-problems intact, in order to avoid falling into more complicated conflicts. Technically, this is equivalent to attribute the variable in the most complex sub-problem to the nogood conclusion, whenever a conflict has to be justified, or a nogood resolution occurs due to a dead-end. The usefulness of variable ordering heuristics (VOHs) appears in these two situations.

The second improvement of EPDB is to satisfy the current constraint C_i before selecting another one to verify. In fact, when a checked constraint C_i is violated, PDB [11] changes the value of a C_i concerned variable and another constraint is checked. The satisfaction of C_i is not insured, but it will be re-checked later.

To reuse solution and past reasoning techniques, EPDB (Figure 1) starts the search with a DCSP that contains a full assignment (e.g. past solution) keeping retroactive data structures of nogoods ngList saved during the past search process.

EPDB updates the set of constraints cstrs which need to be verified and satisfied; it clears the safety conditions list SC, which contains the old orders expressed by deleted nogoods, and initiates the information of the solution obtainment and the empty nogood at False. Then, it checks the consistency of every constraint till finding a solution or generating the empty nogood (lines 1–7).

If an inconsistency is detected, a new nogood ng is generated using *GenerateNg* (lines 7–9). The function is called to determine the variable, among the variables in conflicts *conflictVars*, which will represent the right-hand side (*Rhs*) of the nogood. It generates the nogood respecting the existing order, which is expressed by both of the list of nogoods *ngList* and safety conditions *SC* (lines 30–32), or according to the chosen variable ordering heuristic *VOH* if no order exists (line 33). Next, the new nogood is added into the list of nogoods *ngList*, and *Repair*, the recursive function, is called, in order to find a new assignment to *Rhs*(*ng*), the right-hand side variable (the conclusion) of *ng* (lines 10 and 11).

If the constraint c is consistent, it is moved from the *DCSP* constraints set *cstrs* to *freeCstrs*, which saves all checked and satisfied constraints of the *DCSP*. Then, another constraint is chosen to be checked (lines 12–14). If all constraints are satisfied (*DCSP.cstrs* is empty), the solution is found, and the search is ended (lines 15–17).

As already said, the function *Repair* looks for an assignment to its parameter var, therefore, it verifies its domain (line 18). If the function cannot assign a value to var due to a domain wipe out, it builds a nogood *ng2*, which is the resolution nogood of all nogoods where var appears in the conclusion, using the procedure *ResolutionNg* (lines 18 and 19). If the resolved nogood is empty, no solution can be found, thus, the search is ended (lines 20–22). Else, *Repair* is called recursively to look for a new assignment to *Rhs*(*ng2*), the right-hand side of the nogood *ng2*. If *Rhs*(*ng2*) cannot change its value due to a domain wipe out, an empty nogood is generated. The function returns False due to the solution's non-existence (lines 23 and 24). Otherwise, Repair deletes all nogoods where var exists in the antecedent and updates the safety conditions list *SC* (lines 25 and 26). Updating *SC* consists of deleting from it all orders where the right-hand side is equal to the right-hand side of any of deleted nogoods, before preserving orders imposed by them in *SC*. Then, it updates the set of constraints by moving to *cstrs* the constraints involving var and associates to it a new value from its valid domain (lines 27–28).

ResolutionNg is called when the variable var detects a dead-end. Thus, the procedure stores, in a structure *conflictVars*, all variables appearing on the left-

Algorithm 1 Description of *EPDB*

```
Procedure EPDB(DCSP, newCstrs, VOH)
1.  add newCstrs in DCSP.cstrs;
2.  clear DCSP.SC;
3.  solutionFound ← False;
4.  emptyNgGenerated ← False;
5.  c ← a constraint from DCSP.cstrs;
6.  while (¬solutionFound && ¬emptyNgGenerated) do
7.    if (c is not consistent) then
8.      conflictVars ← variables of c;
9.      ng ← GenerateNg(conflictVars);
10.     add ng in DCSP.ngList;
11.     Repair(Rhs(ng));
12.   else
13.     move c from DCSP.cstrs to DCSP.freeCstrs;
14.     c ← a constraint from DCSP.cstrs;
15.     if (c is null) then
16.       solutionFound ← True;
17. Return Result;

Function Repair(var)
18. if (domain of var is empty) then
19.   ng2 ← ResolutionNg(var);
20.   if (ng2 is empty) then
21.     emptyNgGenerated ← True;
22.     Return False;
23.   if (¬Repair(Rhs(ng2))) then
24.     Return False;
25. remove from DCSP.ngList nogoods as var in Lhs;
26. update DCSP.SC according to deleted nogoods;
27. move var's constraints from DCSP.freeCstrs to DCSP.cstrs;
28. change value of var;
29. Return True;

Function GenerateNg(conflictVars)
30. lower ← get lower priority in conflictVars;
31. if(lower is not null) then
32.   Return nogood between conflictVars where lower is Rhs;
33. Return nogood between conflictVars according VOH;

Function ResolutionNg(var)
34. conflictVars ← variables in Lhs of nogoods prohibiting values of var;
35. GenerateNg(conflictVars);
```

hand side (*Lhs*) of all nogoods prohibiting var's values (line 34). Then, it calls *GenerateNg* to generate the relevant nogood (line 35).

2.3 Profound Degree Heuristic

When a constraint is violated, a concerned variable has to change the value. This change can affect neighbors, hence the interest of deg heuristic to choose the variable to change.

In fact, when a variable changes the value, the change may affect, in addition to its neighbors, all of the network variables, with a given probability.

The goal of profound degree (pdeg) [1] is to change, when a conflict is detected, the concerned variable that can affect the network the least possible. Therefore, pdeg of each variable estimates its influence relatively to its position in the network. This is performed at the beginning of the search. The influence estimation considers the whole network constraints, and associates to each constraint a specific weight, higher or lower, in relation to its distance from the concerned variable. pdeg is considered as the sum of these weights.

3 LAZY REPAIRING BACKTRACKING

3.1 Motivation

EPDB has the advantage to be largely flexible compared to PDB: it exploits the flexibility of PDB to build optimal orders between variables, using VOHs, in order to converge to the termination with minimal checks of constraints.

The drawback of both EBDP and PDB is the verification of the current order between concerned variables before generating a nogood, in conflict or domain wipe-out situations. This verification of orders, which are saved in safety conditions and nogoods, is necessary to avoid looping, but it often consumes a significant time, since in most cases, orders are not directly deduced.

To clarify, suppose the next example:

EPDB is used to repair a dynamic CSP. The current constraint to check is C_{12}, which is violated. A nogood concerning X_1 and X_2 must be generated, thus, the current order between X_1 and X_2 is verified.

The actual orders are shown in Table 1.

Safety Conditions	Nogoods
$X_2 < X_i$ (1)	$X_k = v_k \Longrightarrow X_l \neq v_l$ (4)
$X_i < X_m$ (2)	$X_k = v_k \Longrightarrow X_1 \neq v_1$ (5)
$X_j < X_k$ (3)	$X_i = v_i \Longrightarrow X_j \neq v_j$ (6)

Table 1. Existing orders

In fact, $X_2 < X_1$, since $X_2 < X_i$ (1), $X_i < X_j$ (6), $X_j < X_k$ (3) and $X_k < X_1$ (5). But deducing the order between X_1 and X_2 is not directly done, as the variables

involved in this order, namely X_2, X_i, X_j, X_k and X_1, are involved in other nogoods and safety conditions which are stored before the concerned orders. X_i is involved in (2) which is placed before (6) and X_k in (4) which is before (5), then, (2) and (4) are also verified.

Moreover, determining order takes an expensive time also when no order exists between variables in conflict. Unfortunately, this must be proven by verifying all nogoods and safety conditions concerning them directly (orders where they appear) and indirectly (orders where their neighbours appear directly and indirectly).

As an example, suppose once again using EPDB to repair a DCSP, such as the current constraint to check C_{12} is violated. To generate a nogood, the current order between X_1 and X_2 is verified.

The actual orders are shown in Table 2.

Safety Conditions	Nogoods
$X_2 < X_i$ (1)	$X_k = v_k \Longrightarrow X_l \neq v_l$ (4)
$X_i < X_m$ (2)	$X_1 = v_1 \Longrightarrow X_k \neq v_k$ (5)
$X_j < X_n$ (3)	

Table 2. Existing orders

The approach checks (1) then (2) to test if X_2 precedes X_1, and (5) then (4) to test if X_1 precedes X_2, before concluding that no order exists between them, and establishing one using the specified VOH.

Therefore, in Lazy Repairing Backtracking (LRB), orders are deduced directly using a VOH. Thus, while safety conditions in EPDB are only used to store orders expressed by the removed nogoods, and nogoods are also used to save justifications of prohibitions (to avoid choosing a value which will cause an already met conflict and to restore values when causes are no longer valid), safety conditions are rejected and only nogoods are kept.

The only issue that can exist, when establishing the order directly using a VOH, i.e. without verifying the existence of order between concerned variables, is the termination when the VOHs values change. To remedy this problem, only static VOHs are used, in order to have a fixed order between variables during the search process, especially since the static VOHs, deg and pdeg, have the best behavior in repairing [1, 2].

3.2 Behavior of LRB Compared to PDB and EPDB

As shown in Figure 1, after checking a constraint C_{ij}, satisfied or not, PDB checks another one C_{kl}. In fact, if C_{ij} is violated, PDB tests if an order already exists between the concerned variables i and j. Next, it builds a nogood ng, respecting the order, if there exist. If no order exists between i and j, it builds any nogood ng which will impose a new order between i and j. Then, the conclusion of ng changes the value and another constraint C_{kl} is checked.

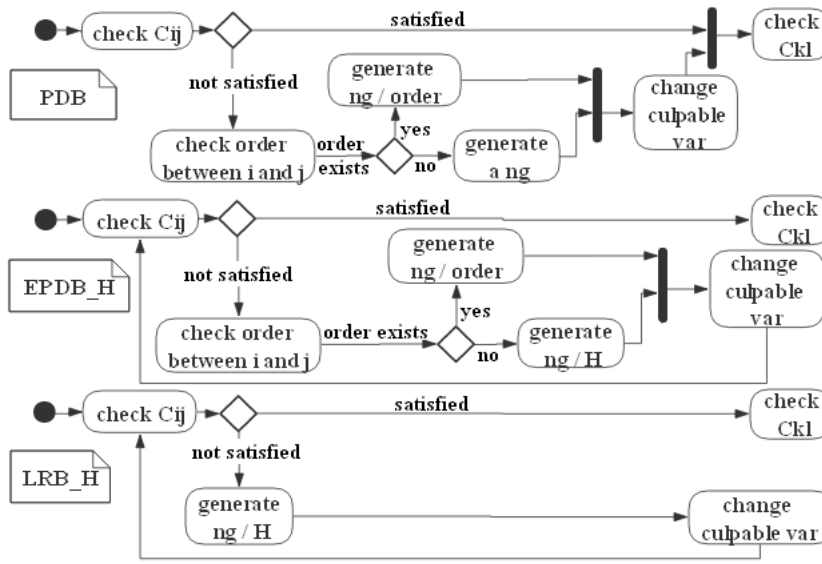


Figure 1. Behaviour of PDB, EPDB and LRB

After checking a constraint C_{ij}, EPDB checks another one C_{kl} only if C_{ij} is satisfied. If C_{ij} is violated, it tests if an order already exists between the concerned variables i and j. Next, it builds a nogood ng, respecting the order if there exists. If no order exists between i and j, the nogood respects the chosen heuristic H. Then, the conclusion of ng changes the value, and C_{ij} is re-checked.

LRB, like EPDB, after checking a constraint C_{ij}, checks another one C_{kl} only if C_{ij} is satisfied. Otherwise, LRB builds a nogood ng, respecting the chosen heuristic H. Then, the conclusion of ng changes the value, and C_{ij} is checked again.

3.3 An Example Run

Let us consider the CSP illustrated in Figure 2:

- $X = \{X1, X2, X3, X4\}$.
- $D = \{D_1, D_2, D_3, D_4\}$ such as $D_i = \{1, 2, 3\}$, $\forall i \in [1, 4]$.
- $C = \{X1 < X2, X1 \neq X4, X2 = X3, X2 \neq X4\}$.

All variables are assigned by the value 1. For simplicity, we consider that this first assignment (before perturbation) was found using a simple resolution approach (FC for example), then, at the beginning of repairing, no nogood or safety condition are saved in the last search. Thus, no order is imposed between variables.

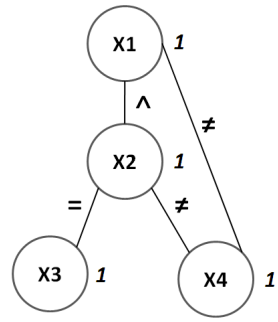


Figure 2. A constraint network example

Table 3 describes EPDB using the deg heuristic. The approach calculates, before starting the search, deg of each variable in the network: $\deg(X1) = 2$, $\deg(X2) = 3$, $\deg(X3) = 1$ and $\deg(X4) = 2$.

All variables have the same assigned value 1. This assignment violates more than one constraint. A constraint ordering heuristic [12] can be used to improve the search, but, in order to treat different cases, we assume that the constraints are ordered as shown in *to check* column.

The first checked constraint *checked* is the one between $X1$ and $X2$ (line 1). This constraint is violated, and as no order exists yet between $X1$ and $X2$ (*add* column is empty), the heuristic value of $X1$ and $X2$ is verified to determine the order. $\deg(X1) < \deg(X2)$, thus, the generated nogood is (1). $X1$ changes the value and the constraint is re-checked before selecting another one (line 2). C_{12} is not consistent, then, the existence of order between $X1$ and $X2$ is tested before generating the nogood. (1) is tested then (2) is generated. $X1$ changes the value and the constraint, which is still not consistent, is re-checked (line 3). To generate the nogood, the existence of order between $X1$ and $X2$ is tested. Then, (1) is tested and (3) is generated. $X1$ has no more values in its domain, then, EPDB resolves (1), (2) and (3) as (4), removes the three no valid nogoods (line 4), saves their expressed order as a safety condition (5) and changes the value of $X2$, the unique cause of the domain wipeout of $X1$ (line 5). $X1$ is assigned to its first valid value, and the constraint, which is consistent, is rechecked (line 6). As C_{12} is satisfied, it is moved to the list of free constraints *free cstrs*, which contains all tested and satisfied constraints, and the next constraint to verify is checked. C_{14} is violated (line 7). The existence of order between $X1$ and $X4$ is tested, but no order in *add* contains neither $X1$ nor $X4$. Then, the heuristic value is verified. As $\deg(X1) = \deg(X4)$, any order is chosen, we assume that in similar cases, the chosen order is the lexicographic one. Thus, the nogood is generated as (6). $X4$ changes the value, then, C_{14} is consistent (line 8) and it is moved to *free cstrs*. The next constraint to check is C_{24} which is violated (line 9), (5) and (6) are tested, then, (7) is generated, the value of $X4$ is changed (line 10) and C_{14}, the only constraint

in *free cstrs* concerning $X4$, is moved to *to check* list. The next checked constraint is C_{23} which is violated (line 11), the tested orders are (5) then (6), and (7). As no order exists between the two variables, the chosen heuristic is used to generate (8), then, the value of $X3$ is changed (line 12). C_{23} is consistent, thus, it is moved to *free cstrs*. C_{14} is then checked (line 13), it is satisfied so it is moved to *free cstrs*. *to check* is empty (line 14), thus, all constraints are satisfied and the solution is found.

	to check	**free cstrs**	$\mathbf{X_1\ X_2\ X_3\ X_4}$	**checked**	**test**	**add**	**remove**
1	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		1, 1, 1, 1	$C_{12}\times$	deg	$X_2{=}1 \Rightarrow X_1 \neq 1$ (1)	
2	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		2, 1, 1, 1	$C_{12}\times$	(1)	$X_2{=}1 \Rightarrow X_1 \neq 2$ (2)	
3	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		3, 1, 1, 1	$C_{12}\times$	(1)	$X_2{=}1 \Rightarrow X_1 \neq 3$ (3)	
4	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		–, 1, 1, 1			$X_2 \neq 1$ (4)	(1), (2), (3)
5	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		–, 2, 1, 1			$X_2 < X_1$ (5)	
6	$C_{12}\,C_{14}\,C_{24}\,C_{23}$		1, 2, 1, 1	$C_{12}\,\surd$			
7	$C_{14}\,C_{24}\,C_{23}$	C_{12}	1, 2, 1, 1	$C_{14}\times$	deg	$X_1{=}1 \Rightarrow X_4 \neq 1$ (6)	
8	$C_{14}\,C_{24}\,C_{23}$	C_{12}	1, 2, 1, 2	$C_{14}\surd$			
9	$C_{24}\,C_{23}$	$C_{12}\,C_{14}$	1, 2, 1, 2	$C_{24}\times$	(5),(6)	$X_2{=}2 \Rightarrow X_4 \neq 2$ (7)	
10	$C_{24}\,C_{23}\,C_{14}$	C_{12}	1, 2, 1, 3	$C_{24}\surd$			
11	$C_{23}\,C_{14}$	$C_{12}\ C_{24}$	1, 2, 1, 3	$C_{23}\times$	(5),(6),(7),deg	$X_2{=}2 \Rightarrow X_3 \neq 1$ (8)	
12	$C_{23}\,C_{14}$	$C_{12}\ C_{24}$	1, 2, 2, 3	$C_{23}\surd$			
13	C_{14}	$C_{12}\ C_{24}\,C_{23}$	1, 2, 2, 3	$C_{14}\surd$			
14		$C_{12}\ C_{24}\,C_{23}\,C_{14}$	1, 2, 2, 3	NULL			

Table 3. Execution of $EPDB_{deg}$

Similarly to EPDB, when LRB is executed using the same heuristic deg, the approach calculates, before starting the search, deg of all network variables.

It proceeds like EPDB with the following differences:

- to generate a nogood, LRB tests directly the chosen VOH values. Then, in test column, each constraint violation (nogood to generate) causes only one test, which is the test of deg, contrary to EPDB (the test column of Table 3).
- LRB does not need the information about the order saved in safety conditions, since the order is determined by the used VOH. Then, when a nogood is removed, no safety condition is retained (in Table 3, after removing nogoods in remove column of line 4, add column of line 5 will be empty).

3.4 Description of the Proposed Approach

To reuse the solution and the past reasoning techniques, LRB starts the search with a DCSP that contains the past solution, keeping retroactive data structures of nogoods ngList saved during the past search process.

LRB proceeds exactly like EPDB, with the following major differences:

- to generate a nogood when a conflict is detected, LRB does not verify the order between concerned variables as EPDB does. Then, in line 9 of Figure 1, the chosen VOH is directly used to establish the order.
- similarly, to generate the resolution nogood when a domain wipe-out is produced, LRB does not verify the order between the responsible variables as EPDB does.

Thus, in line 35 of Figure 1, the chosen VOH is directly used to generate the nogood.

- As no order verification will be required, LRB does not use the safety condition list to store orders expressed by the removed nogoods, then, lines 2 and 26 in Figure 1 are removed.

3.5 Termination Proof

It is proven in [6] that PDB terminates. In fact, PDB respects the relative ordering conditions corresponding to the ordered nogoods. Moreover, it imposes safety conditions, which are relative ordering conditions, in order to retain some of the ordering conditions of the nogoods that have been removed. The order selected for new nogoods is required to respect both types of ordering conditions. Thus, the resulting set of conditions could not be cyclic.

LRB proceeds in the same manner of PDB, but drops safety conditions. To guarantee termination, it uses only static VOHs and is based on the used SVOH for the selected order.

4 EXPERIMENTAL STUDY

4.1 Experiments Concerning Repairing

In this section, we compare LRB to EPDB, to show the improvement of the new approach. We also compare it to MAC-2001 [4, 5], to identify the maximal rate of perturbation that tolerates repairing, i.e. from which rate of disturbance the use of a good resolution algorithm is more efficient to deal DCSPs.

Some integrations of MAC in intelligent backtrackers were already proposed [16, 13], but we choose MAC-2001 as we judge it to be one of the most efficient resolving CSPs strategies while being the most familiar constraint propagation one.

Heuristics used in LRB and EPDB are pdeg and deg. To prove that performance of LRB is due essentially to a well-chosen heuristic, we use also the lexicographic order heuristic, which is used instead of the arbitrary order, as LRB supports only the static ordering to avoid looping. MAC-2001 is combined to the dom/deg heuristic like in [14]. We note that the arc consistency strategy can be also improved by using dom/wdeg [7], but as shown in [15], this heuristic is efficient essentially in combination with AC-3.

As evaluation criteria, we measure the runtime and the number of Constraints Checks (CCs).

4.1.1 Experiments on Uniform Random Binary DCSP

A uniform random binary DCSP P is a sequence of static CSPs which are subject to perturbations.

CSPs are characterized by $\langle n, d, p_1, p_2 \rangle$, where n is the number of variables, d the number of values per variable, p_1 the network connectivity, defined as the ratio of existing binary constraints to possible binary constraints, and p_2 the constraints tightness, that represents the proportion of forbidden pairs among the constraints.

Tests are performed on sparse and dense problems, respectively $\langle 30, 10, 0.25, 0.45 \rangle$ and $\langle 30, 10, 0.75, 0.19 \rangle$. The chosen problems are the satisfiable ones at the transition phase [8] (the most complicated area).

For each pair of fixed density and constraints tightness $\langle p_1, p_2 \rangle$, 30 different instances are treated by each algorithm and the present results are the averages of these 30 runs.

The rate of injected constraints $\%IC = \frac{|new\ \ constraints|}{|whole\ \ problem\ \ constraints|}$. It indicates the rate of disturbance that will be treated using both LRB and EPDB. IC varies from 1 % to 100 %, such as for each given disturbance of d %, the added constraints are the same as those in the disturbance of $d - 1$ %, plus 1 % more constraints. The 30 final problems (after disturbance) are the same for each rate of disturbance.

For DCSP approaches, namely LRB and EPDB, the computed effort is the one for repairing the past solution, i.e. which existed before disturbing.

As p_2 is known, we tried to replace, in the pdeg calculation, the parameter 1/2 by p_2. But results were not significantly better than those of using 1/2, then, we choose to keep the parameter 1/2 for all kinds of problems. We fix the time-out at 7 minutes. Experiments are performed on the Java platform, on a core i7 PC (3.6 GHz processor and 16 GB RAM).

Concerning the number of CCs (2^{nd} part of Tables 4 and 5), using LRB with a VOH is almost the same as using EPDB with the same VOH. But regarding the execution time (1^{st} part of Tables 4 and 5), LRB exceeds EPDB very remarkably, regardless of the used VOH.

Excepting the small disturbances, MAC-2001 often produces less CCs than LRB (2^{nd} part of Tables 4 and 5), mainly when LRB uses the deg heuristic.

From a certain small disturbance (13 % for sparse problems and 10 % for those dense), LRB needs almost the same effort to repair (Tables 4 and 5), whatever the rate of injected constraints IC. This effort is not stable, regardless of the used heuristic, but it is subject to same sudden falls.

Regarding LRB, the use of pdeg is often more beneficial than using deg.

LRB_{pdeg} is always faster than MAC-2001 (1^{st} part of Tables 4 and 5), even disturbing 100 % of constraints, which is equivalent to resolving the whole problems. LRB_{deg} is sometimes less efficient than MAC-2001, but it is anyway a good concurrent.

4.1.2 Experiments on Meetings Scheduling Problems

A Dynamic Meeting Scheduling Problem (DMSP) is a sequence of static Meeting Scheduling Problems (MSPs): MSP_0, MSP_1, ..., $MSP_{\alpha-1}$, MSP_α, ..., where the network is subject to constraints perturbations, and each MSP_i differs from the latest one MSP_{i-1} by the addition of some constraints.

% IC	MAC-2001	LRB_{pdeg}	LRB_{deg}	LRB_{lex}	$EPDB_{pdeg}$	$EPDB_{deg}$	
1		**0.006**	0.008	1.471	0.508	0.723	
4		**0.024**	0.026	1.835	2.242	2.662	
7		**0.048**	0.065	2.047	4.842	6.254	
10		**0.059**	0.073	2.304	5.788	7.047	
13		**0.079**	0.089	2.650	7.517	8.373	
16		**0.082**	0.093	2.860	8.081	9.338	
19		**0.082**	0.097	2.809	8.099	9.243	
22		**0.085**	0.094	2.876	8.214	9.479	
25		**0.080**	0.095	2.878	7.759	9.479	
28		**0.080**	0.099	2.860	8.020	9.467	
31		**0.078**	0.101	2.891	7.263	8.093	
34	0.094	**0.075**	0.102	2.881	7.330	8.402	**time(sc)**
40		**0.062**	0.075	2.869	8.321	8.069	
46		**0.069**	0.078	2.893	9.521	10.455	
52		**0.080**	0.093	2.894	8.919	10.799	
58		**0.065**	0.076	2.914	8.897	8.989	
64		**0.075**	0.084	2.942	8.593	8.258	
70		**0.082**	0.097	2.932	7.678	9.113	
76		**0.073**	0.106	2.871	8.189	10.227	
82		**0.085**	0.099	2.903	8.266	10.381	
88		**0.083**	0.100	2.889	8.203	10.337	
94		**0.085**	0.099	2.897	8.510	10.315	
100		**0.084**	0.099	2.868	8.230	9.888	
1		**15 108**	20 199	3 032 339	**15 108**	20 199	
4	169 168	**58 651**	65 202	3 806 153	59 779	68 687	
7		**117 728**	157 599	4 227 876	123 667	157 298	
10		**145 623**	181 242	4 764 235	148 286	180 593	
13		192 355	214 418	5 527 584	189 534	209 805	
16		201 644	225 471	5 992 181	205 562	230 519	
19		201 099	235 586	5 879 701	204 280	228 941	
22	**169168**	207 509	230 595	6 049 455	206 419	233 566	
25		196 193	233 567	6 047 661	197 054	235 723	
28		196 130	243 182	6 001 166	203 289	237 828	
31		191 143	247 404	6 060 702	185 955	206 190	
34		183 019	248 225	6 058 750	188 308	211 733	**CCs**
40	169 168	**151 765**	185 843	6 015 178	212 178	202 816	
46	**169 168**	170 804	192 706	6 066 024	242 173	261 430	
52		197 833	226 562	6 076 485	228 440	268 696	
58	169 168	**161 864**	187 689	6 094 090	227 370	228 662	
64		186 754	207 362	6 138 907	223 674	210 593	
70		200 774	237 995	6 029 206	195 718	229 670	
76		181 101	255 407	6 029 857	213 139	257 020	
82	**169 168**	207 836	246 870	6 084 857	210 158	259 925	
88		201 640	248 743	6 070 954	210 870	259 146	
94		207 763	243 764	6 068 184	217 752	257 468	
100		206 479	241 355	6 014 275	207 955	246 269	

Table 4. Execution time by seconds and Number of CCs ($p_1 = 0.25$, $p_2 = 0.45$)

% IC	MAC-2001	LRB_{pdeg}	LRB_{deg}	LRB_{lex}	$EPDB_{pdeg}$	$EPDB_{deg}$	
1		**0.698**	0.896	1.874	111.769	194.669	
4		**3.328**	4.565	13.510	370.385	–	
7		**4.698**	5.877	32.571	–	–	
10		**6.590**	7.427	40.478	–	–	
13		7.379	**6.675**	42.993	–	–	
16		7.657	**7.591**	44.719	–	–	
19		**7.847**	8.638	44.873	–	–	
22		**8.835**	9.253	44.903	–	–	
25		**5.962**	8.694	44.027	–	–	
28		**7.711**	9.874	44.113	–	–	
31		**7.848**	9.780	44.086	–	–	
34	11.054	**8.938**	10.503	44.004	–	–	**time(sc)**
40		**8.357**	9.736	43.991	–	–	
46		**8.183**	9.297	44.067	–	–	
52		10.209	**9.597**	44.148	–	–	
58		**9.427**	10.268	43.885	–	–	
64		10.794	**10.731**	43.960	–	–	
70		**10.688**	11.700	43.985	–	–	
76		**8.355**	9.925	44.026	–	–	
82		**8.946**	10.461	44.056	–	–	
88		**8.271**	9.475	44.145	–	–	
94		**7.615**	8.883	44.184	–	–	
100		**7.600**	8.898	44.242	–	–	
1	11 091 737	**2 053 499**	2 676 894	5 300 103	2 063 315	2 687 746	
4		10 050 253	13 609 155	37 787 281	**7 700 192**	–	
7		14 196 226	17 492 480	90 474 816	–	–	
10		19 917 644	22 168 200	112 775 270	–	–	
13		22 323 569	19 950 505	119 366 418	–	–	
16		23 178 909	22 661 403	121 865 776	–	–	
19		23 692 559	25 854 188	122 555 689	–	–	
22		26 684 225	27 586 331	122 734 624	–	–	
25		18 061 593	25 937 447	122 755 174	–	–	
28		23 355 364	29 466 473	122 739 959	–	–	
31		23 793 094	29 203 775	122 837 687	–	–	
34		27 114 111	31 384 172	122 669 346	–	–	**CCs**
40	**11 091 737**	25 268 174	29 127 000	122 734 902	–	–	
46		24 710 542	27 748 892	122 643 912	–	–	
52		30 820 375	28 720 750	122 943 597	–	–	
58		28 428 460	30 704 619	122 704 450	–	–	
64		32 524 541	32 018 965	122 798 424	–	–	
70		32 150 097	34 963 393	122 669 749	–	–	
76		25 192 875	29 655 806	122 768 330	–	–	
82		26 922 873	31 354 768	122 696 907	–	–	
88		24 982 344	28 307 565	122 684 444	–	–	
94		22 954 833	26 581 585	122 766 266	–	–	
100		22 965 095	26 579 493	122 848 545	–	–	

Table 5. Execution time by seconds and Number of CCs ($p_1 = 0.75$, $p_2 = 0.19$)

A Meeting Scheduling Problem MSP is characterized by $\langle m, p, n, d, h, t, a\rangle$, where m is the number of meetings, p the number of participants, n the number of meetings per participant and d the number of days. Different time slots are available for each meeting, and h is the number of hours per day, t the duration of the meeting and a the percentage of availability for each participant.

In the used model, $p \times n$ is the number of variables. Variables $X_{i.j}$, such as $i \in p$, $j \in n$ and $X_{i.j} \in [timeSlot(1), timeSlot(m)]$, are the totality of meetings per all participants. Thus, tow types of constraints exist:

- As p is the number of participants and n the number of meetings per participant, for each two distinct variables $X_{i.j}$ and $X_{k.l}$, if $i = k$, then, variables are connected by a constraint termed *Arrival Constraint*, which means that the participant i must attend two distinct meetings. Therefore, $|X_{i.j} - X_{k.l}| \geq \delta$, where δ is the time required to move between $X_{i.j}$ and $X_{k.l}$, expressed in time slots.
- For each two distinct participants i and k, $X_{i.j} = X_{k.l}$ means that i and k must attend the same meeting.

We present our results for the class $\langle 15, 30, 4, 5, 10, 1, 85\rangle$ and vary the rate of disturbance from $\sim 1\,\%$ to $\sim 100\,\%$. We generated 30 different instances treated by each algorithm and the results are an averages of these 30 runs. As LRB_{lex} and EPDB are not good concurrent, as shown in the experiments above, we include in comparison only MAC-2001, LRB_{pdeg} and LRB_{deg}.

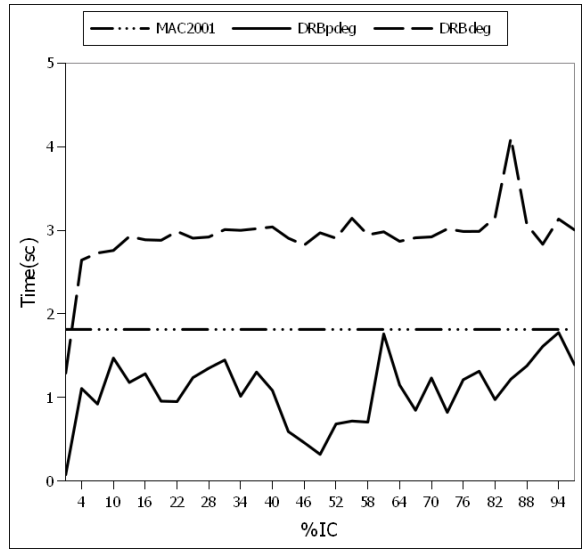
Concerning MSPs (Figure 3), LRB_{pdeg} is more efficient than MAC-2001 regardless of the rate of disturbance, specially respecting the number of CCs. However, MAC-2001 is more effective than LRB_{deg} respecting runtime (except for a disturbance of $1\,\%$), even if the rate of CCs is comparable to that of LRB.

4.2 Experiments Concerning the Problem Solving

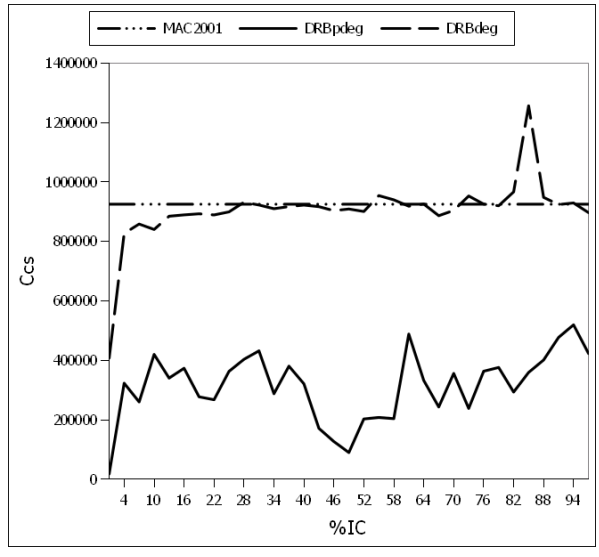
The previous experiments concern satisfiable problems. Experiments on random problems are more precisely focused on satisfiable problems in hard areas of low and high densities. These experiments show that LRB using a VOH, mainly pdeg, is a good concurrent for MAC-2001 although for solving ($100\,\%$ of IC).

In order to show where MAC-2001 is more efficient than LRB_{pdeg} and LRB_{deg}, we execute the three approaches on random problems, not necessary satisfiable, with the same previous parameters n, d and p_1. Constraints tightness' vary from $p_2 = 0.1$ to $p_2 = 0.9$. 30 different instances are treated by each algorithm and the present results are the averages of these 30 runs. Experiments are performed on the *Java* platform, on a Pentium IV PC (2.8 GHz processor and 3.24 GB RAM).

Figures 4 and 5 show that LRB, using either deg or pdeg, is efficient to treat satisfiable problems, i.e. $\langle p_1 = 0.25, p_2 \in [0.1, 0.45[\rangle$ and $\langle p_1 = 0.75, p_2 \in [0.1, 0.19[\rangle$, while MAC-2001 is effective to detect the non satisfiability of problems in $\langle p_1 = 0.25, p_2 \in]0.45, 0.9]\rangle$ and $\langle p_1 = 0.75, p_2 \in]0.19, 0.9]\rangle$. In complex areas, i.e. the

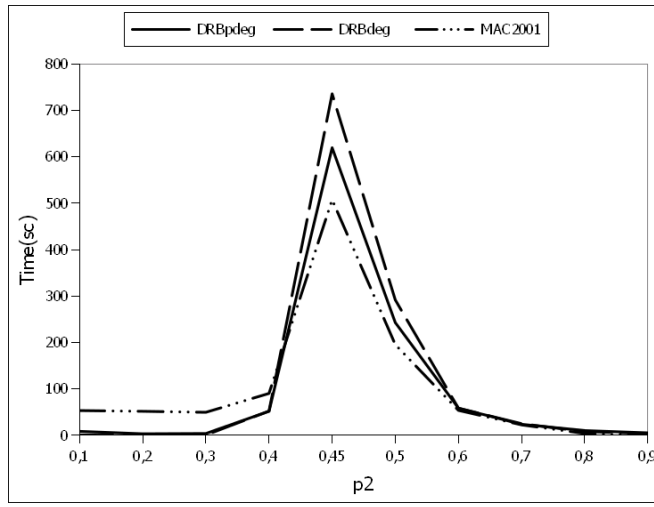


a)

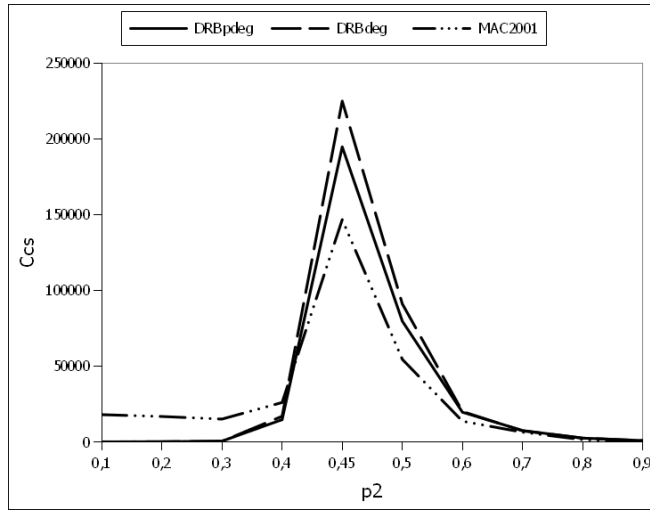


b)

Figure 3. Execution time and Number of CCs for MSPs

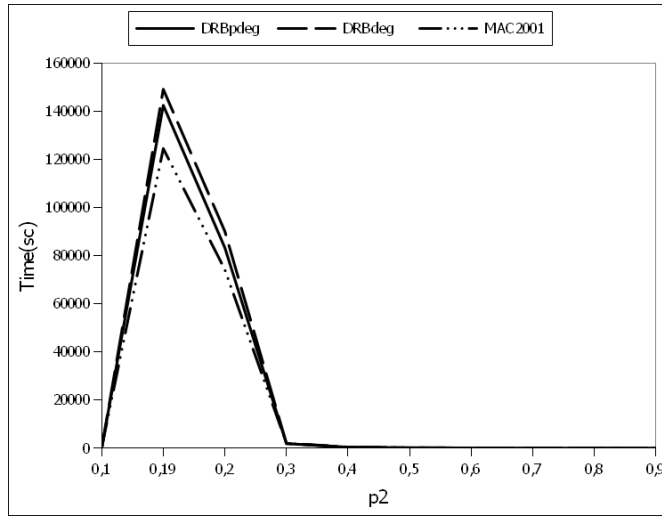


a)

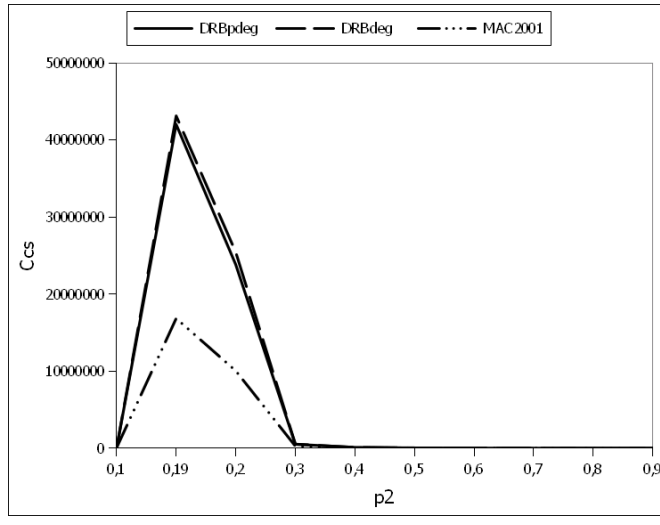


b)

Figure 4. Execution time and Number of CCs for $p_1 = 0.25$



a)



b)

Figure 5. Execution time and Number of CCs for $p_1 = 0.75$

transition phases $\langle p_1 = 0.25, p_2 = 0.45\rangle$ and $\langle p_1 = 0.75, p_2 = 0.19\rangle$, the majority of problems are non satisfiable, therefore, MAC-2001 is generally better than LRB, mainly for high densities, where satisfiable problems are rare.

For high density problems (Figure 5), using either deg or pdeg in LRB is almost the same.

5 DISCUSSION

5.1 Discussion Concerning Repairing

As shown in the experiments on satisfiable problems in the transition phases (Tables 4 and 5), LRB is widely more powerful than EPDB in terms of runtime. This is particularly due to avoiding the verification of progressively saved orders, which is done in EPDB each time inconsistency is detected. But regarding CCs, the use of LRB with a VOH is almost the same as using EPDB with the same VOH. Normally, for a fixed heuristic, LRB and EPDB proceed in the same manner, with the difference of testing orders for EPDB. Thus, they must generate the same number of CCs. However, the perturbation, which is a modification in constraints, can change the used heuristic values, as both of the used VOHs deg and pdeg depend on the existing constraints. Then, in EPDB, nogoods stored before disturbance impose an order which is not necessary the same as the one expressed by the present heuristic values, whence the difference of the number of CCs between LRB and EPDB.

From a certain small disturbance, LRB requires almost the same effort to repair, whatever the rate of injected constraints, since problems are chosen from the most complex areas (the transition phases). This effort is not stable, but it is subject to the same sudden falls. In fact, the repairing effort depends on the affectation before disturbance and its consistency with the new constraints. Then, disturbing a problem with some constraints can be easier or harder than disturbing another problem with more constraints, depending on which affectation is satisfiable by more new constraints.

The use of pdeg in LRB (Tables 4 and 5 and Figure 3) is often more effective than using deg, as pdeg estimates the position of each variable regarding the whole network, while deg calculates the position of each variable regarding its neighbourhood.

When problems are hard and satisfiable, LRB, mainly when using pdeg, is a good concurrent of MAC-2001, even for solving (100 % of IC). Nevertheless, when no VOH is used (LRB_{lex} in 1^{st} part of Tables 4 and 5), LRB becomes inefficient comparing to MAC-2001. In fact, the goal of VOHs in repairing is to engender minimal network changes, by keeping the hard sub-problems unchangeable as long as possible, in order to minimize risks of affecting other sub-problems. Therefore, the performance of LRB is also due to the use of a good VOH.

5.2 Discussion Concerning Solving

Concerning problems where constraints tightness is less than the peak (under the transition phase), although injecting the totality of constraints in perturbation, i.e. resolving (Figures 4 and 5), LRB using either deg or pdeg is more efficient than MAC-2001, regardless of problems density. In these areas, most of the problems are satisfiable, thus, the use of arc consistency is not efficient enough. As a matter of fact, the major interest of MAC-2001 is to detect earlier the future domains dead-ends, in order to modify the partial assignments which can not be included in a solution. But in sparse graphs, domains dead-ends are rarely met.

Therefore, for unsatisfying problems, where constraints tightness is higher than the peak, MAC-2001 is more effective than LRB.

On peaks, the majority of problems are unsatisfying, then, MAC-2001 is often more efficient than LRB.

Respecting high densities, LRB_{pdeg} does not exceed remarkably LRB_{deg}. This is caused by the high connectivity of the network. Therefore, each variable has the majority of other variables as neighbors, then, pdeg is nearly the same as deg.

6 CONCLUSION AND PERSPECTIVES

In this paper, we introduce a new approach, namely LRB, which is based on EPDB, to repair the DCSPs solutions rapidly, using static VOHs.

Based on the results of experiments on a variety of problem classes, we show that LRB is a great improvement of EPDB, respecting the runtime, insofar as it avoids the expensive tests of orders.

We have proved that this approach is mainly efficient when using pdeg and that is a good concurrent for MAC-2001. In fact, LRB outperforms MAC-2001 when problems are satisfiable, although using it to solve from the scratch. Thus, LRB has the advantage to be not only destined for DCSPs, but also to CSPs.

We demonstrate that LRB is more effective to treat satisfiable problems while MAC-2001 is more efficient to detect the problem's non-satisfiability. Therefore, for complicated problems where the satisfiability is unknown, the two approaches can be separately launched and the fast resolution is then guaranteed.

As perspectives, we plan to implement a CSP solver, based on LRB, which benefits from LRB advantages and resolves by repairing a complete assignment that violates a minimum number of constraints, in order to converge more rapidly to a solution. We also plan to develop a version of LRB for minimal perturbation problems [19], to preserve DCSPs solutions as much as possible. We also intend to integrate constraint propagation within LRB to benefit from the advantage of the two strategies regardless of the problem's satisfiability.

REFERENCES

[1] Benamrane, A.—Acodad, Y.—Benelallam, I.—Bouyakhf, E. H.—Bennani Othmani, M.: Modeling Trainings in Faculty of Medicine and Pharmacy of Casablanca as Constraint Satisfaction Problem. The Thirteenth International Workshop on Constraint Modelling and Reformulation, 2014, pp. 17–23.

[2] Zhang, W.—Xing, Z.—Wang, G.—Wittenburg, L.: An Analysis and Application of Distributed Constraint Satisfaction and Optimization Algorithms in Sensor Networks. Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '03), Vol. 3, 2003, pp. 185–192, doi: 10.1145/860575.860605.

[3] Verfaillie, G.—Jussien, N.: Constraint Solving in Uncertain and Dynamic Environments: A Survey. Constraints, Vol. 10, 2005, No. 3, pp. 253–281, doi: 10.1007/s10601-005-2239-9.

[4] Dechter, R.—Dechter, A.: Belief Maintenance in Dynamic Constraint Networks. Proceedings of the Seventh AAAI National Conference on Artificial Intelligence (AAAI '88), 1988, pp. 37–42.

[5] Ginsberg, M.—McAllester, D.: GSAT and Dynamic Backtracking. In: Borning, A. (Ed.): Principles and Practice of Constraint Programming (PPCP 1994). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 874, 1994, pp. 243–265, doi: 10.1007/3-540-58601-6_105.

[6] Acodad, Y.—Benelallam, I.—Hammoujan, S.—Bouyakhf, E. H.: Extended Partial-Order Dynamic Backtracking Algorithm for Dynamically Changed Environments. 2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI), Vol. 1, 2012, pp. 580–587, doi: 10.1109/ICTAI.2012.83.

[7] Acodad, Y.—Benamrane, A.—Benelallam, I.—Bouyakhf, E. H.: Profound Degree: A Conservative Heuristic to Repair Dynamic CSPs. In: Iliadis, L., Maglogiannis, I., Papadopoulos, H. (Eds.): Artificial Intelligence Applications and Innovations (AIAI 2014). Springer, Berlin, Heidelberg, IFIP Advances in Information and Communication Technology, Vol. 436, 2014, pp. 140–149, doi: 10.1007/978-3-662-44654-6_14.

[8] Bessière, C.—Régin, J.-C.: Refining the Basic Constraint Propagation Algorithm. Proceedings of the 17th International Joint Conference on Artificial Intelligence, Vol. 1, 2001, pp. 309–315.

[9] Bessière, C.—Régin, J.-C.—Yap, R. H. C.—Zhang, Y.: An Optimal Coarse-Grained Arc Consistency Algorithm. Artificial Intelligence, Vol. 165, 2005, pp. 165–185, 2005, doi: 10.1016/j.artint.2005.02.004.

[10] Ginsberg, M. L.: Dynamic Backtracking. Journal of Artificial Intelligence Research, Vol. 1, 1993, No. 1, pp. 25–46, doi: 10.1613/jair.1.

[11] Hammoujan, S.—Acodad, Y.—Benelallam, I.—Bouyakhf, E. H.: Dynamic Constraint Ordering Heuristics for Dynamic CSPs. Applied Mathematical Sciences, Vol. 7, 2013, No. 138, pp. 6889–6907, doi: 10.12988/ams.2013.311657.

[12] Bliek, C.: Generalizing Partial Order and Dynamic Backtracking. Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence (AAAI'98/IAAI'98), 1998, pp. 319–325.

[13] Prosser, P.: MAC-CBJ: Maintaining Arc Consistency with Conflict-Directed Backjumping. Research Report 95/177, Department of Computer Science, University of Strathclyde, Glasgow, 1995.

[14] Jussien, N.—Debruyne, R.—Boizumault, P.: Maintaining Arc-Consistency within Dynamic Backtracking. In: Dechter, R. (Ed.): Principles and Practice of Constraint Programming – CP 2000. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1894, 2000, pp. 249–261, doi: 10.1007/3-540-45349-0_19.

[15] Mehta, D.—van Dongen, M. R. C.: Reducing Checks and Revisions in Coarse-Grained MAC Algorithms. Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05), 2005, pp. 236–241.

[16] Boussemart, F.—Hemery, F.—Lecoutre, C.—Sais, L.: Boosting Systematic Search by Weighting Constraints. Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04), IOS Press, 2004, pp. 146–150.

[17] Mehta, D.—Van Dongen, M. R. C.: Probabilistic Consistency Boosts MAC and SAC. Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), 2007, pp. 143–148.

[18] Crawford, J. M.—Auton, L. D.: Experimental Results on the Crossover Point in Random 3-SAT. Artificial Intelligence, Vol. 81, 1996, No. 1, pp. 31–57, doi: 10.1016/0004-3702(95)00046-1.

[19] Zivan, R.—Grubshtein, A.—Meisels, A.: Hybrid Search for Minimal Perturbation in Dynamic CSPs. Constraints, Vol. 16, 2011, No. 3, pp. 228–249, doi: 10.1007/s10601-011-9108-5.

Yosra Acodad graduated in 2018 from the Mohammed V University. Her research interests focused on the repair algorithms of satisfaction dynamic constraints problems, one of the popular paradigms of complexe problem-solving at LIMIARF Laboratory under the supervision of Professor El Houssine Bouyakhf and Professor Imade Benelallam. She has published many papers in various international conferences and journals.

Amine Benamrane is Ph.D. student at the Mohammed V University. His research interests focused on the repair algorithms of satisfaction distributed dynamic constraints problems, a subfield of artificial intelligence. He received his M.Sc. in computer science from the Mohammed V University in Rabat in 2009. Since 2009, he is an executive at the Scientific Research and Innovation Department of the Higher Education Ministry of Morocco. His work is focused on promoting of research results and boosting collaborations between researchers and companies. In 2014 he began his Ph.D. at LIMIARF Laboratory under the supervision of Professor El Houssine Bouyakhf and Professor Imade Benelallam. He has published papers in various international conferences and journals.

El Houssine Bouyakhf graduated from Supaéro Toulouse, and he obtained his doctorate in artificial vision at the LAAS Laboratory of the CNRS in Toulouse. Avant-garde and visionary, he was convinced very early on that AI technologies would generate breakthrough innovations and a plethora of business applications thus founded LIMIARF in 1998, the first African laboratory for AI and robotics in Rabat, Morocco. He coordinated more than 40 doctoral students there, collaborated with several industrial companies and published 120 research papers in international journals and conferences. Deeply interested in the transfer of knowledge from the scientific ecosystem to the business world and feeling that more and more European and African companies are adopting data technologies, he finally founded AIOX Labs at the advent of the new era of AI. As CEO of the first spin-off company of AI laboratories in Morocco, he coordinates the development of all AI projects and products to adapt the results of his research to the needs of companies. He is also proud to create jobs for his former doctoral students that allow them to demonstrate their true values and skills.

Imade Benelallam is Associate Professor, teaching data-driven courses, big data, artificial intelligence, computer science and insight data engineering at the National Institute of Statistics and Applied Economics in Rabat and other international engineering schools and universities. He is Research Director of SI2M Laboratory and Coordinator of data science engineering degree at the INSEA. He earned his Masters degrees from FS Rabat and INPT in computer science and telecom and EMI in industrial electronics. He received his Ph.D. degree in artificial intelligence and computer science from the Mohammed the Fifth University in April 2010. His Ph.D. research focused on constraint programming and distributed constraint reasoning. His contributions aim to propose and combine new models and algorithms in constraints technology using learning for intelligent and ethical agents. He co-founded AOIX LABS, the first Artificial Intelligence startup Laboratory spin-off in Morocco. He is Active Volunteer in the IEEE association since 2004. Currently, he is appointed as IEEE Senior Member, an IBM Certified Instructor and Oracle Academy Instructor. He works in different projects within industrial partners as IT Consultant or Research Fellow; the most significant was with Thales group. He has authored several scientific publications and e-books in many conferences and journals. He served as reviewer for international journals and conferences such as Applied Soft Computing Elsevier etc.