Profound Degree: A Conservative Heuristic to Repair Dynamic CSPs

Yosra Acodad, Amine Benamrane, Imade Benelallam, and El Houssine Bouyakhf

[1] LIMIARF. Faculty of Sciences Mohammed the Fifth University - Agdal Rabat, Morocco
{yosra.acodad,benamraneamine}@gmail.com, bouyakhf@fsr.ac.ma
[2] INSEA, National Institut of Statistics and Applied Economic - Irfane Rabat, Morocco
imade.benelallam@ieee.org

Abstract. For a better treatment of Dynamic Constraint Satisfaction Problems ($DCSPs$), several techniques have been developed to be used in repair algorithms. We cite, for example, the variables/values ordering heuristics and local search techniques.

We distinguish between static heuristics, which calculate their values once at the beginning of the search, and dynamic heuristics that use an expensive intelligence in terms of solving time.

In this paper, we propose a new static variable ordering heuristic, Profound Degree ($pdeg$), based on deg heuristic, which calculates the degree of influence of a given variable, on the whole constraints network, relatively to its position in the network.

We evaluate this heuristic on the Extended Partial-order Dynamic Backtracking ($EPBD$) approach, which is an approach to repair $DCSPs$ solutions, and we compare it to the best-known variables ordering heuristics ($VOHs$) for repairing. The evaluation of performance is on random binary problems and meeting scheduling problems, with the criteria of computation time, number of constraints checks and Hamming distance between the former and the current solution.

1 Introduction

In recent years, several improvements have been made in algorithms for solving and repairing $DCSPs$ [3] and [4]. Among these, we quote $VOHs$ [9], filtering techniques and conflicts analysis.

In fact, the order in which variables are affected by a search algorithm is crucial. Indeed, the use of different $VOHs$ to solve the same CSP can lead to very different results in terms of performance.

The used heuristics can be classified into two broad categories: Static variables ordering heuristics ($SVOHs$) and dynamic variables ordering heuristics ($DVOHs$).

In this paper, we present a new $SVHO$, *profound degree heuristic* ($pdeg$), which is inspired by the deg heuristic insofar as it uses the concept of neighborhood.

L. Iliadis et al. (Eds.): AIAI 2014, IFIP AICT 436, pp. 140–149, 2014.

In effect, *pdeg* is not limited to calculate the number of neighbor connections of concerned variable, but it also measures the impact of its eventual disturbances (since we study this heuristic under correction of $DCSPs$) in relation to all constraints network, i.e., it takes into consideration all levels of the network neighborhoods, by giving each level a different weight, then it keeps intact the value of the variable having a strong impact, in order to reduce disturbing the network.

2 Background

2.1 Minimal Perturbation Problem (MPP)

Definition 1 (MPP). *A Minimal Perturbation Problem (MPP) is a triple $\Pi = (\Theta, \alpha, \delta)$, where:*

- *Θ is a CSP.*
- *α is a partial or complete assignment for Π that is called initial assignment.*
- *δ is a function that defines a distance between any two assignments.*

A solution to a MPP is a solution to Π with minimal distance from α according to δ.

Definition 2 (Distance Set D). *Let σ and γ be partial assignments for Θ. V is the set of variables and $W(\sigma, \gamma)$ is the set of variables such that the value assignment for v in σ is different from the value assignment for v in γ :*

$$W(\sigma, \gamma) = \{v \in V \mid \langle v, val\rangle \in \sigma, \langle v, val'\rangle \in \gamma, val \neq val'\} \tag{1}$$

$W(\sigma, \gamma)$ is called a distance set for σ and γ and the elements of the set are called perturbations.

Definition 3 (Function δ). *In an MPP, the distance function of some assignment σ from α is defined as the cardinal of the set $D(\sigma, \alpha)$:*

$$\delta(\sigma, \alpha) = \mid D(\sigma, \alpha) \mid \tag{2}$$

Otherwise, δ is the hamming distance between σ and α.

Definition 4 (Solution Value Assignment (SVA)). *α is the initial assignment of a CSP. For each variable v_i:*

$$If \langle v_i, val_{i_j}\rangle \in \alpha \Longrightarrow v_i.SVA = val_{i_j} \tag{3}$$

SVA is the acronym of Solution Value Assignment.

2.2 Extended Partial-Order Dynamic Backtracking (*EPDB*)

Unlike several *CSP* resolution approaches that use a fixed order of variables, the Partial-order Dynamic Backtracking (*PDB*) [2] uses a partial order, that is built from nogoods and safety conditions, that is dynamically modified during the search process.

Definition 5 (Nogood). *A nogood is a failure justification. It's an expression of the form :*

$$(x_1 = v_1) \wedge ... \wedge (x_k = v_k) \rightarrow x \neq v$$

Definition 6 (Safety Conditions). *A safety condition 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 antisymmetric.

$$x <_s y \Longrightarrow x \leq_s y, y \nleq_s x \tag{4}$$

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

$$x < z_1 < z_2 < ... < z_n < y \tag{5}$$

When having to generate a nogood during the search, due to conflict situation (creating a nogood caused by a violated constraint) or domain wipe out (nogoods resolution), and when no order is imposed in existing nogoods and safety conditions between variables concerned with this conflict, there may be significant choices for the variable that will emerge as the generated nogood conclusion. The performance of this approach depends mainly on the quality of this choice, which is not covered in *PDB*.

The major goal of *EPDB* is to extend the classic description of *PDB*, by exploiting its flexibility to repair assignments using repair-directed heuristics, when having to build an order between some variables, to control changes.

3 Profound Degree Heuristic (*pdeg*)

3.1 Description of Heuristic

The *deg VOH* [5] has a major advantage, it's that it does not compute every time dynamic information about the current state of research (such as the current domains size *dom* or the current variables degree *ddeg* [10]), but it operates static information.

However, in the context of repairing solutions, this heuristic appears limited. In fact, no information regarding the position of variables in relation to the entire network is operated, whereby the change of a variable can affect not only the neighbors, but all of the variables in the network with a given probability. For this, we propose to calculate an estimation of this information for each variable

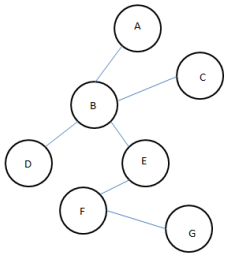

Fig. 1. A simple constraints network

X at the beginning of the research, considering all the network constraints, and by associating to each constraint a weight, higher or lower, in relation to its distance from X. *pdeg* is considered as the sum of those weights.

Now consider the simple network above (figure 1). Suppose that each variable has a consistent value and due to a perturbation of the $DCSP$, the value of E must be changed. Assume that all constraints have the same tightness p_2. Changing E will generate the following probabilities P:

$$P(violate(C_{EB})) = P(violate(C_{EF})) = p_2$$
$$P(violate(C_{BA})) = P(violate(C_{BC})) = P(violate(C_{BD})) = P(violate(C_{FG})) = p_2 * p_2 = {p_2}^2$$

If the probabilities of violating constraints are considered as weights, knowing that $p_2 \in [0, 1]$, the weights of constraints network are:

$$Weight(C_{EB})_{/E} = Weight(C_{EF})_{/E} = p_2 \geqslant Weight(C_{BA})_{/E} = Weight(C_{BC})_{/E} = Weight(C_{BD})_{/E} = Weight(C_{FG})_{/E} = {p_2}^2$$

This reflects that, whenever the constraint C is closer to the variable X, C is more influenced by the change of X, since the variables concerned by C are more influenced by changing X.

In fact, given two constraints C_1 and C_2, if the constraint C_1 is closer to X than C_2, then C_1 has a cost in calculation of the heuristic value of X which is more interesting than C_2, since changing X engenders a risk of violating C_1 more important than violating C_2.

In reality, constraints networks are not as simple as that, they are quite as figure 2:

On the other hand, the hardness of constraints network (especially when the constraints used in the modeling are intent) is not always known, reasons why choosing to replace the weight, that corresponds to the tightness of the constraints, by an average hardness of 1/2.

Inspired by this idea, *pdeg* heuristic accumulates, for each variable X, the weights of all constraints. These weights are higher each time the constraints are closer to X.

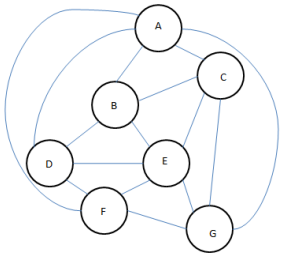

Fig. 2. A constraints network

Function PDEG(X,CSP)
1. $pdeg \leftarrow 0$;
2. $level \leftarrow 0$;
3. $size \leftarrow CSP.Vars.size()$;
4. $actual \leftarrow X$;
5. $total \leftarrow X$;
6. **while**($totat.size() != size$)
7. $\quad neighbors \leftarrow getAllNeighbors(actual)$;
8. $\quad filter(neighbors, total)$;
9. $\quad actual \leftarrow getDistinct(neighbors)$;
10. $\quad a \leftarrow neighbors.size()$;
11. $\quad b \leftarrow nbConstraints(actual)$;
12. $\quad total.add(actual)$;
13. $\quad pdeg += 1/2^{incLevel()} \times a + 1/2^{incLevel()} \times b$;
14. *Return* $pdeg$;

Fig. 3. Description of *pdeg* heuristic

3.2 Execution of Heuristic

Both of the value of *pdeg* and the *level*, that is incremented each time the constraint is far from the variable in question, are initialized by 0 (lines 1 and 2). The variable *size* contains the number of variables in the CSP (line 3) and *actual* and *total* lists, initiated by the variable for which having to calculate the *pdeg*, serve respectively to store all variables in the current level and all variables already involved in the calculation (lines 4 and 5).

Will not accounting all constraints network variables, the heuristic calculates the neighborhood of the current variables, then this neighborhood is filtered to include only new variables, to not record a same constraint twice (lines 6-8).

Note that $getAllNeighbors()$ (line 7) recognizes a neighbor many times as the number of constraints that bind it with the current variables.

The heuristic puts into *actual* current neighbors eliminating redundancies, in a the number of neighbors of actual variables with redundancy (which is equivalent to the number of constraints with neighbors) not yet recognized and in b the number of constraints between these neighbors (lines 9 and 11). At this point, it adds to *total* the current neighbors and to *pdeg* the weight of neighbors, which is 1/2 to the power the *level* incremented, multiplied by the number of

redundant neighbors, and it does the same thing for the number of constraints between neighbors, after incrementing the *level* (lines 12 and 13).

The heuristic leaves the while loop once all variables are accounted, which means it counted all the constraints of the network, and the value of *pdeg* is returned as the profound degree of the variable X (line 14).

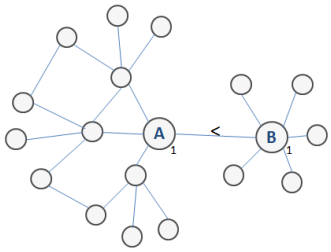

Fig. 4. Graph of constraints

3.3 Example of Simulation

We consider the constraints network above and assume that all variables have a consistent value ($A = 1$, $B = 1$ and all other variables have values consistent with the whole constraints network). We suppose that the $DCSP$ has undergone a disturbance that is the change of the nature of the constraint C_{AB}, that was: $A = B$, and that became: $A < B$. To satisfy this constraint, the $EPDB$ algorithm [1] is restarted in order to repair solution. $EPDB$ chooses to change the value of A, or the one of B, depending on the used heuristic.

Using $EPDB_{deg}$, heuristic values are:

$deg(A) = 4$

$deg(B) = 6$.

So the approach generates the *nogood*: $B = 1 \Longrightarrow A \neq 1$, since it retains the value of the most relevant variable in the network, i.e., the one whose change can disrupt more the $DCSP$ and delay resolution. For this reason, the value of A is changed.

Now, using $EPDB_{pdeg}$, the values of used heuristic are:

$pdeg(A) = 1/2 \times 4 + 1/2^2 \times 1 + 1/2^3 \times 14 + 1/2^4 \times 2 = 4,125$

$pdeg(B) = 1/2 \times 6 + 1/2^2 \times 0 + 1/2^3 \times 3 + 1/2^4 \times 1 + 1/2^5 \times 9 + 1/2^6 \times 2 = 3,75$

Then the approach generates the *nogood*: $A = 1 \Longrightarrow B \neq 1$ and changes the value of B.

4 Experimental Results

We carried out a series of experimental tests to compare the integration of *pdeg* heuristic, within the $EPDB$ algorithm, to other $VOHs$.

Heuristics used for repairing are those that improve the best the behavior of $EPDB$, namely the degree (deg), the domain size (dom), the number of conflicts ($conf$) and the ratio between the domain size and the weighted degree ($dom/wdeg$) [6].

We evaluate the performance in terms of constraints checking (CCs), computing time ($Time(sc)$) and Hamming distance (HD) [8] and [7].

All experiments were performed on the Java platform.

4.1 Experiments on Randoms

Random $CSPs$ are characterized by a quadruplet $< n, d, p_1, p_2 >$, where n is the number of variables, d the number of values per variable, p_1 the network connectivity (density) and p_2 the constraints tightness.

The density p_1 of a constraints graph is defined as the the ratio of the number of constraints relative to the maximum number of possible constraints in this network.

The tightness p_2 of a constraint corresponds either to the proportion of unauthorized tuples, or the probability that a tuple is not allowed.

Tests have been performed on sparse and dense problems, respectively $< 20, 10, 0.25, 0.6 >$ and $< 20, 10, 0.75, 0.27 >$. The constraints tightness is selected such as areas are as complex as possible while problems have solution.

For logical reasons, we assume that the maximum rate of added constraints is equivalent to $\sim$25%. For each pair, 20 instances were solved using each heuristic, and the results are presented as an average of these 20 instances.

Figure 5 presents the effectiveness of heuristics applied to the $EPBD$, for the most complex regions of the low density selected. In terms of number of constraints tested CCs, Between 10% and 24% of new injected constraints, $pdeg$ is the best, elsewhere, $dom/wdeg$ and dom are the most efficient. Otherwise, the deg is the closest to $pdeg$, which seems the most stable.

In time graph, $dom/wdeg$ recedes (also for dom but not in the same manner), due to the calculation of dom and $wdeg$ heuristics, especially for $wged$. $pdeg$ is almost always the best.

Concerning the Hamming distance HD, there is a competition between heuristics, and $pdeg$ remains a good competitor in most regions.

Figure 6 shows the effectiveness of the heuristics for the most complex areas of the high density p_1=0.75. In terms of number of CCs and $execution\ time$, $pdeg$ usually looks the best although not improving too much the behavior of deg. Concerning Hamming distance, no heuristic is absolutely the best.

4.2 Experiments on Dynamic Meeting Scheduling Problems

A Meeting Scheduling Problem MSP is characterized by $< m, p, n, d, h, t, a >$, 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.

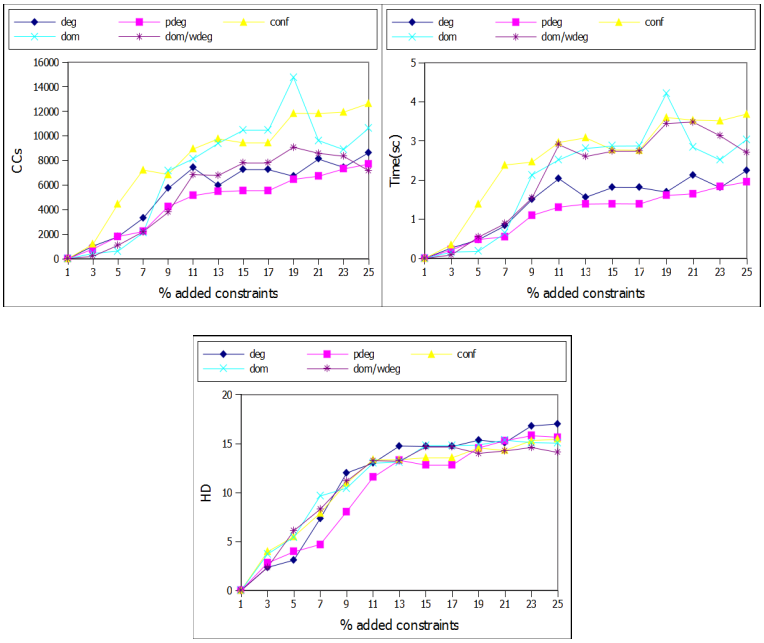

Fig. 5. Number of CCs, Execution time performed and HD (p_1=0.25, p_2=0.60)

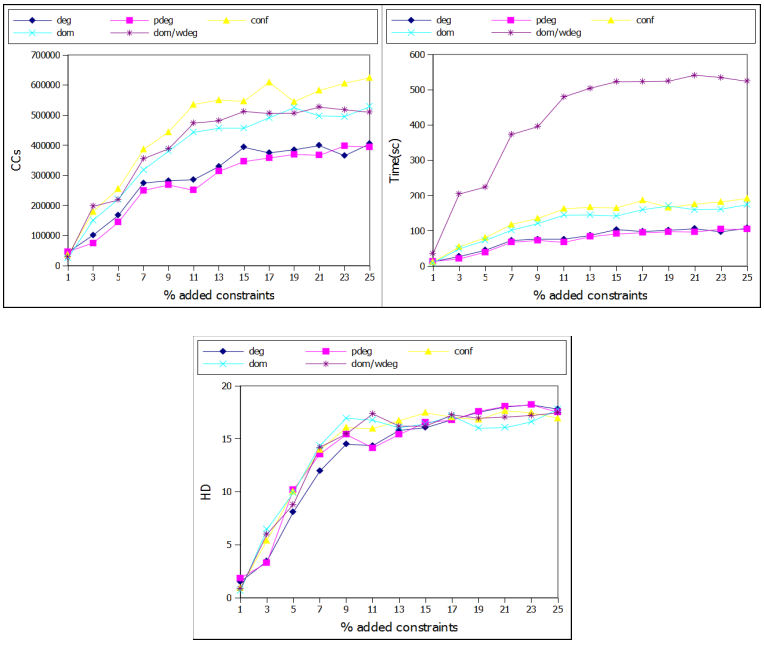

Fig. 6. Number of CCs, Execution time performed and HD (p_1=0.75, p_2=0.27)

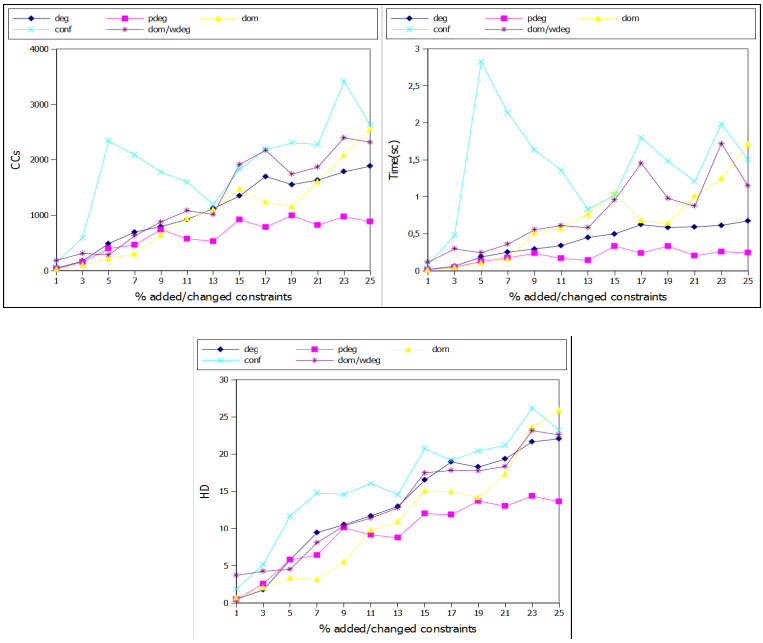

Fig. 7. Number of CCs, Execution time performed and HD

We present our results for the class $< 20, 5, 15, 5, 10, 1, 70 >$ and we vary the rate of changed constraints from ∼1% to ∼25%. We generated 20 different instances solved using each heuristic in $EPDB$.

Figure 7 shows the effectiveness of the heuristics in terms of number of CCs, *execution time* and HD. *pdeg* leads to a great improvement in most areas (from 10% of disrupted constraints in terms of CCs, 7% in terms of *time* and 11% in terms of HD).

4.3 Discussion

For low densities, *pdeg* behaves better than *deg*, more remarkably than high densities. In fact, in high densities, almost all variables are connected, i.e., for each given variable, almost all other variables in the network are its neighbors. Therefore calculating *pdeg* of a variable almost returns to calculate its *deg*.

pdeg leads, especially for low densities, to a minimum Hamming distance (HD). In fact, the idea of *pdeg* is to change the value of the least constraining variable in the network, i.e., the one that, when changing its value, will cause the least change to other variables. And knowing that the variables before the correction have $SVAs$ values, so a minimum change of network variables is a minimum change of $SVAs$, then a maximum of $SVAs$.

Although the heuristics *dom*, $conf$ and $dom/wdeg$, in some regions, do less constraints checking than static heuristics *deg* and *pdeg*, they consume more computational time because of their dynamic aspect.

5 Conclusion

In this paper, we introduce a new VOH, *pdeg* heuristic, for estimating the influence of variables in the whole constraint network, in the process of repairing solutions in dynamic environments.

Based on the results of experiments on a wide class of problems, we found and proved that this heuristic, used in $DCSPs$ correction, exceeds the heuristics deemed to be efficient when solving $CSPs$.

In fact, the idea of *pdeg* heuristic is to disturb only the parts of the constraints network, whose probability of influence is minimal. Indeed, for low to medium densities, as seen in the random problems with density $p_1 = 0,25$ and the MSP (and especially for medium densities), using *pdeg* improves remarkably the repair of solution, not only in relation to research performance (*time* and CCs), but also with respect to the quality of the solution (*Hamming distance*).

As perspectives, we intend to integrate this heuristic in other algorithms for solving and repairing centralized and distributed $CSPs$, to hybridize it with other heuristics according to areas in which they show a remarkable improvement in order to lead to an optimal heuristic, and to experiment it on real problems.

References

1. Acodad, Y., Benelallam, I., Hammoujan, S., Bouyakhf, E.H.: Extended Partial-order Dynamic Backtracking algorithm for dynamically changed environments. In: 2012 IEEE 24th International Conference on (ICTAI), pp. 580–587 (November 7, 2012)
2. Ginsberg, M.L., McAllester, D.A.: Gsat and dynamic backtracking. Journal of Artificial Intelligence Research 1, 25–46 (1994)
3. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: AAAI, pp. 37–42 (1988)
4. Bessiere, C.: Arc-consistency in dynamic constraint satisfaction problems. In: Proc. AAAI 1991, pp. 221–226. AAAI Press (1991)
5. Dechter, R., Meiri, I.: Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In: Proceedings of IJCAI 1989, pp. 271–277 (1989)
6. Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI (August 2004)
7. Zivan, R., Alon, G., Amnon, M.: Hybrid search for minimal perturbation in Dynamic CSPs. Constraints 16(3), 228–249 (2011)
8. Hebrard, E., Barry, O., Walsh, T.: Distance Constraints in Constraint Satisfaction. In: IJCAI (January 6, 2007)
9. Ortiz-Bayliss, J., Terashima-Marin, H., Ender, O., Andrew, J., Santiago, E.: Exploring heuristic interactions in constraint satisfaction problems: A closer look at the hyper-heuristic space. In: 2013 IEEE Congress on Evolutionary Computation (CEC). IEEE (June 20, 2013)
10. Bessiere, C., Jean-Charles, R.: MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 61–75. Springer, Heidelberg (1996)