Boosting systematic search by weighting constraints

Frédéric Boussemart and **Fred Hemery** and **Christophe Lecoutre** and **Lakhdar Sais** [1]

Abstract. In this paper, we present a dynamic and adaptive variable ordering heuristic which guides systematic search toward inconsistent or hard parts of a Constraint Satisfaction Problem (CSP). This generic heuristic is able to exploit information about previous states of the search process whereas traditional dynamic ones only exploit information about the current state. Intuitively, it avoids some trashing by first instantiating variables involved in the constraints that have frequently participated in dead-end situations. Such information is recorded by associating a weight with each constraint. This weight is increased whenever the associated constraint is violated during search. The extensive experiments that we have conducted prove that our conflict-directed approach is the most efficient current one with respect to significant and large classes of academic, random and real-world instances.

1 Introduction

In recent years, many improvements to backtracking algorithms for solving constraint satisfaction problems (CSPs) have been proposed. Roughly speaking, such improvements mainly concern ordering heuristics, filtering techniques, and conflict analysis. They can be conveniently classified as look-ahead and look-back schemes [9]. Look-ahead schemes are used when extending the current partial solution by maintaining, for example, a given level of local consistency, in order to anticipate future conflicts. Look-back schemes are designed to learn from conflicts in order to avoid the same conflict situations occurring later in the search. Based on experimental analysis, these two paradigms are usually considered as not entirely orthogonal. More precisely, the enhancement of look-ahead techniques is sometimes counterproductive to the effects of look-back techniques [4, 8]. We agree to some extent on such observation. In fact, in the general case, the difficulty of conceiving a good variable ordering heuristic which avoids redundant search makes learning from conflicts necessary.

In this paper, we propose a fruitful combination of look-back and look-ahead schemes. Our main objective is to address more efficiently structured problems, where some constraints are more important than others, and consequently some parts of the problem are inconsistent and/or more hard to solve. To this end, we introduce a dynamic and adaptive heuristic, denoted $wdeg$, which is able to direct systematic search to inconsistent or hard parts of a CSP. Our approach is able to learn (look-back side) and then to exploit (look-ahead side) information from previous states of the search process whereas traditional static and dynamic heuristics only exploit information about the initial and current state of the search, respectively. More precisely, inspired from works [21, 23, 20, 26, 7] about different methods developed for SAT (satisfiability testing) and CSP, a higher weight is given to constraints violated at some previous states of the search process. Such weighted constraints are then used by $wdeg$ in order to guide a backtrack search-like algorithm. In other words, as search progresses, the weight of hard constraints become more and more important and this particularly helps the heuristic to select variables appearing in the hard part of the CSP. In contrast with sophisticated look-back schemes (see e.g. [9]), that use complex conflict analysis techniques and heavy data structures, our approach can be grafted in a very simple way, to a backtrack search algorithm.

Extensive experiments prove that our approach is the most efficient current one wrt significant and large classes of academic, random and real world instances. As expected, the improvements are very huge when addressing problems with (local) inconsistent subparts. Surprisingly enough, significant improvements are also obtained on others classes of instances such as random and academic ones, showing the robustness and the efficiency of our approach.

The paper is organized as follows. First, some technical background about constraint satisfaction is recalled. Next, after a brief overview about variable ordering heuristics, our approach is motivated and described. Some results of our experimentation on large and significant classes of CSPs instances are then introduced. Before concluding, some related works are presented.

2 Definitions and preliminaries

In this section, we briefly introduce some notations and definitions used hereafter.

Definition 1 *A Constraint Satisfaction Problem (CSP) is a pair* $(\mathscr{X}, \mathscr{C})$ *where:*

- $\mathscr{X} = \{X_1, \ldots, X_n\}$ *is a finite set of* n *variables such that each variable* X_i *has an associated domain* $dom(X_i)$ *denoting the set of values allowed for* X_i,
- $\mathscr{C} = \{C_1, \ldots, C_m\}$ *is a finite set of* m *constraints such that each constraint* C_j *has an associated relation* $rel(C_j)$ *denoting the set of tuples allowed for the variables* $vars(C_j)$ *involved in the constraint* C_j.

We shall say that a constraint C binds (or involves) a variable X if and only if X belongs to $vars(C)$. The arity of a constraint C is the number of variables bound by C. The degree of a variable X is the number of constraints where it is involved. Two variables X_i and X_j are said to be neighbors if there exists a constraint C such that $\{X_i, X_j\} \in vars(C)$. We define $\Gamma(X_i)$ as the set of variables that are neighbors of X_i.

A solution is an assignment of values to all the variables such that all the constraints are satisfied. A problem is said to be satisfiable (or consistent) iff it admits a solution, and unsatisfiable otherwise. Solving a CSP P involves either determining the unsatisfiability of P or

[1] CRIL-CNRS FRE 2499, rue de l'université, SP 16, 62307 Lens cedex, France. email: {boussemart,hemery,lecoutre,sais}@cril.univ-artois.fr

finding one (or more) solution. A depth-first search algorithm with backtracking can then be applied, where at each step of the search, a variable assignment is performed followed by a filtering process called constraint propagation. Usually, constraint propagation algorithms, which are based on some constraint network properties such as arc-consistency, remove some values which can not occur in any solution. The algorithm that maintains arc consistency during search is called MAC.

3 Weighting constraints from conflicts

In this section, after a brief review of existing variable ordering heuristics, we introduce an illustrative example that shows the importance of recording constraint violations in order to efficiently circumscribe inconsistent or hard parts of CSPs. Next, we propose a new heuristic that relies on dynamic constraint weighting in order to direct search toward the most constrained part of the CSP.

3.1 An overview of variable ordering heuristics

The order in which variables are assigned by a backtracking search algorithm has been recognized as a key issue for a long time. Using different variable ordering heuristics to solve a CSP can lead to drastically different results in terms of efficiency. Furthermore, we know that simply introducing some kind of randomization to a given variable ordering heuristic may exhibit a large variability in performance [14]. In fact, one should ideally use a variable ordering that selects first a small (strong) backdoor [27], i.e., a set of variables which once assigned makes the resulting problem easy to solve.

A first category of heuristics corresponds to static (or fixed) variable ordering heuristics (SVOs) which are heuristics that keep the same ordering all along the search, and, hence, only exploit (structural) information about the initial state of the search. Such heuristics are $lexico$ where variables are lexicographically ordered, deg and $ddeg$ where variables are decreasingly ordered according to their initial [10] and current degree, $width$ where variables are ordered in order to minimize the width of the constraint graph [11].

A second category of heuristics corresponds to dynamic variable ordering heuristics (DVOs) which take into account some information about the current state of the search. A well-known dynamic heuristic is dom [16] where variables are increasingly ordered according to the current size of their domains. The justification of this heuristic is given [16] by the fail-first principle : "To succeed, try first where you are most likely to fail". When combining domain sizes and variables degrees, one obtains dom/deg [4] and $dom/ddeg$ [4, 25] which can substantially improve the performance of the search. Other theory-based dynamic heuristics have been proposed by [13, 17] but these heuristics, although conceptually elegant, require extra computation and have only been tested on random problems.

It is important to note that ties can occur when using a variable ordering heuristic. A tie is a set of variables that are considered equivalent by the heuristic. We can assume that $lexico$ is always implicitly used to break ties. For instance, dom implicitly corresponds to dom+$lexico$ which selects among the variables with smallest current domain sizes the first variable wrt the lexicographic order. Other tie-breakers can be explicitly introduced (but, if necessary, $lexico$ still remains the last and definitive implicit one). Some new composed heuristics are dom+deg [12], dom+$ddeg$ [5, 24], $BZ3$ [24].

Recently, [3] have proposed a generalization (denoted here $mDVO$) of existing variable ordering heuristics by considering that each variable can be evaluated while taking into account its neighborhood. For instance, assuming that an heuristic h is based on a function α_h which allows evaluating variables, it is possible to define with an operator $\odot$, a generalization $h_1^{\odot}$ of the heuristic h at a neighborhood distance equal to 1. This new heuristic is based on a function $\alpha_{h,1}^{\odot}$ that evaluates any variable X_i as follows:

$$\alpha_{h,1}^{\odot}(X_i) = \frac{\sum_{X_j \in \Gamma(X_i)} (\alpha_h(X_i) \odot \alpha_h(X_j))}{|\Gamma(X_i)|^2}$$

Finally, many experiments have been performed to compare the impact of using different variable ordering heuristics. No heuristic clearly outperforms the other ones but dynamic ones such as dom, dom+$ddeg$ and $dom/ddeg$ are usually considered as the most effective ones. Neighborhood generalization of dom and $dom/ddeg$ have also been shown to be promising [3].

3.2 Circumscribing inconsistent sub-problems

In this subsection, it is shown that recording constraint violations is important to circumscribe inconsistent or hard parts of CSP instances. To give some insight into the main motivation of our approach, we consider, in the following, an example built by merging a consistent CSP with an inconsistent one.

Let us consider the well-known queens problem which involves putting n queens on a chessboard of size $n{*}n$ while avoiding that two queens can attack each other. In a classical CSP model of this problem, each queen is targeted to a row and is represented by a variable whose domain contains exactly n values (corresponding to the different columns of the chessboard), and a constraint is introduced between each pair of variables in order to guarantee that two queens cannot be placed on the same column or diagonal.

Let us introduce another academic problem, called knights problem, that involves putting k knights on a chessboard of size $n{*}n$ such that all knights form a cycle (when considering knight moves). This problem does not admit any solution when the value of k is odd. In a CSP model of this problem, each knight is represented by a variable whose domain contains exactly $n{*}n$ values (corresponding to all squares of the chessboard), and a constraint is introduced between each variable and the variable that comes next (modulo k) in order to guarantee that one can pass from a knight to another with a single knight move. Besides, for any pair of variables, there is a constraint implying that two knights cannot be placed on the same square.

When combining queens and (an odd number of) knights, one can observe an interesting behaviour called thrashing. Here, we simply merge the two independent problems without any interaction. Indeed, the knights subproblem is unsatisfiable but, as classical variable ordering heuristics selects first queens variables (due to their small domain sizes), the unsatisfiable subproblem is rediscovered for each new solution of the queens subproblem.

To observe this phenomenon, we have attached a counter to each constraint and we have run a MAC algorithm using all variable ordering heuristics cited above. Whenever a constraint has been shown unsatisfied (during the constraint propagation process), its attached counter has been incremented by 1.

We have been interested in two values which correspond to the maximum value of counters attached to queens constraints (i.e. constraints involving queens) and knights constraints (i.e. constraints involving knights), respectively. Figure 1 shows, while using $dom/ddeg$, the evolution[2] of these two values with respect to the

[2] A similar behavior can be observed when using other variable ordering heuristics such as dom, $dom \oplus ddeg$, ...

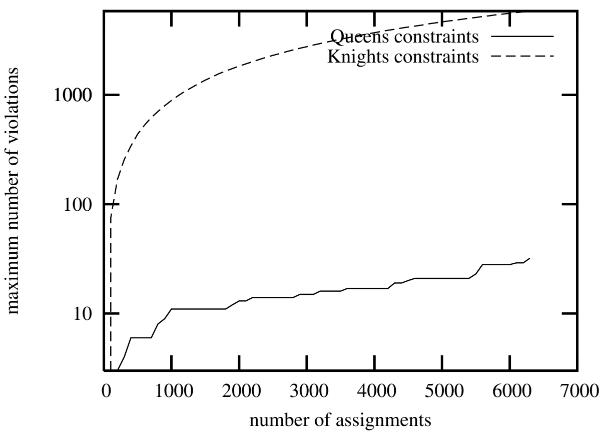

Figure 1. Evolution of constraint violations: 5 knights and 8 queens

number of assignments required to prove the inconsistency of the instance involving 8 queens and 5 knights. It clearly appears that some knights constraints are violated far more often than all queens constraints. in other words, these counters emphasize the inconsistent knights sub-problem.

These results illustrate and confirm that the number of times each constraint is violated during the search is an important information that might be used to locate the inconsistent (hard) part of the CSP. In the following section, we present new heuristics that exploit the weight of the constraints to direct the search to such an important part of the CSP.

3.3 A conflict-directed variable ordering heuristics

As stated in the previous section, traditional dynamic variable ordering heuristics benefit from information about the current state of the search such as current domain sizes and current variable degrees. One limitation of this approach is that no information about previous states of the search is exploited. We propose to capture such information by associating a counter, called $weight$, with any constraint of the problem. These counters will be updated during search whenever a dead-end (domain wipe-out) occurs. As systematic solvers such as FC or MAC involve successive revisions in order to remove values that are no more consistent with the current state, it suffices to introduce a test at the end of each revision. Note that the algorithm 1, presented below wrt coarse-grained filtering algorithms, can be easily adapted wrt fine-grained ones.

More precisely, a general description of the $revise$ function is depicted in Algorithm 1. Note that the function $seekSupport$ determines whether there exists a support of (X, a) in C. According to the implementation of this function, we obtain different coarse-grained filtering algorithms (see, e.g., [19]). At the end of the algorithm (lines 4 and 5), when a domain wipe-out occurs, the weight of the "revising" constraint is incremented.

Algorithm 1 revise(C : Constraint, X : Variable) : boolean

1: **for** each $a \in dom(X)$ **do**
2:     **if** $seekSupport(C, X, a) = false$ **then**
3:         remove a from $dom(X)$
4: **if** $dom(X) = \emptyset$ **then**
5:     $weight[C]$ ++
6: return $Dom(X) \neq \emptyset$

Using these counters, it is possible to define a new variable ordering heuristic, denoted $wdeg$, that gives an evaluation $\alpha_{wdeg}(X_i)$, called weighted degree, of any variable X_i as follows:

$$\alpha_{wdeg}(X_i) = \sum_{C \in \mathscr{C}} weight[C] \mid vars(C) \ni X_i \wedge \mid FutVars(C) \mid > 1$$

where $FutVars(C)$ denotes the uninstantiated variables in $vars(C)$. Hence, the weighted degree of a variable X_i corresponds to the sum of the weights of the constraints involving X_i and at least another uninstantiated variable. Intuitively, locally inconsistent or hard parts of CSPs should be first examined by the search algorithm when selecting in priority variables with greatest weighted degrees, respecting the so-called fail-first principle.

It is important to note that this new heuristic is related to $ddeg$ as only constraints involving a variable and at least another uninstantiated one are considered. In fact, setting all $weight$ counters to 1 is equivalent to define $ddeg$. Then, in order to benefit, at the beginning of the search, from relevant information about current variable degrees, we propose to initialize all $weight$ counters to 1. Finally, combining weighted degrees and domain sizes yields $dom/wdeg$, an heuristic that selects first the variable with the smallest ratio current domain size to current weighted degree. In the rest of the paper, $wdeg$ and $dom/wdeg$ will be called conflict-directed (variable ordering) heuristics.

4 Comparative results

To prove the practical interest of the conflict-directed heuristics introduced in this paper, we have implemented them and performed some experiments with respect to random, academic and real-world problems. Performances for one solution have been measured in terms of the number of constraint checks (#ccks), the number of assignments (#asgs) and the cpu time in seconds (cpu). Also, on some problems involving 50 or 100 instances, the number of solved instances (#solved) is indicated and performance criteria are given on average. We have used a MAC algorithm (called GAC when constraints are non binary) which integrates the coarse-grained arc consistency algorithm $AC3.2$ [19].

4.1 Illustrative example

First, we show that $wdeg$ drastically improves the performances of MAC compared to $dom/ddeg$ with respect to our illustrative example (see subsection 3.2).

Table 1 shows results obtained for two different chessboard sizes ($n = 8$ and $n = 12$) and for three different unsatisfiable problems:

- K_5: the 5-knights instance as defined in subsection 3.2,
- $K_5 \oplus Q_n$: the 5-knights and the n-queens instances merged,
- $K_5 \otimes Q_n$: the 5-knights and the n-queens instances merged such that queens and knights cannot share the same square.

These results clearly show the trashing phenomenon that occurs when using the $dom/ddeg$ heuristic: the number of assignments (and constraint checks) to solve a $K_5 \oplus Q_n$ or a $K_5 \otimes Q_n$ instance is roughly equal to the product of the number of assignments (and constraint checks) to solve K_5 by the number of solutions of the queens instance (92 for n=8, and 14200 for n=12). This behaviour is not observed when using the conflict-directed heuristics. Indeed, after finding a limited number of queens solutions, the knights variables are selected in priority since the weight of the knights constraints become large enough, thereby, avoiding trashing.

n	$instances$		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$
8	K_5	cpu	0.13	0.11	0.1
		#ccks	0.089M	0.083M	0.076M
		#asgs	0.14K	0.09K	0.06K
	$K_5 \oplus Q_8$	cpu	0.33	0.35	3.91
		#ccks	0.395M	0.39M	6.992M
		#asgs	0.5K	0.3K	6.3K
	$K_5 \otimes Q_8$	cpu	0.18	0.62	3.31
		#ccks	0.129M	0.807M	5.376M
		#asgs	0.1K	0.7K	5.7K
12	K_5	cpu	0.35	0.32	0.28
		#ccks	0.436M	0.413M	0.376M
		#asgs	0.3K	0.2K	0.1K
	$K_5 \oplus Q_{12}$	cpu	1.9	4.23	$2,845.18$
		#ccks	3.017M	7.613M	$5,381.667$M
		#asgs	1.8K	3.2K	$2,174.2$K
	$K_5 \otimes Q_{12}$	cpu	0.61	7.89	$2,515.13$
		#ccks	0.617M	12.869M	$4,521.207$M
		#asgs	0.3K	5.5K	$2,017.2$K

Table 1. Knights-Queens instances

4.2 Experiments

Here, we present some representative results obtained from our experimentation. For the real-world (and artificially-generated) binary instances of the fullRLFAP (Radio Link Frequency Assignment Problem) archive, we follow the approach of [3] to produce harder instances by removing some constraints and/or some frequencies. For example, $scen07$-$w1$-$f4$ corresponds to the instance $scen07$ for which the constraints of weight greater than 1 have not been taking into account and the 4 highest frequencies have been removed. Table 2 shows the results obtained on some selected difficult instances. Note that the right-most column gives the best result obtained when using the following four variants of $mDVO$:
$\alpha^{\times}_{dom,1}$, $\alpha^{\times}_{\frac{dom}{ddeg},1}$,$\alpha^{+}_{dom,1}$ and $\alpha^{+}_{\frac{dom}{ddeg},1}$.

The good behavior of conflict-directed heuristics is here clearly shown: using such heuristics allows to solve all instances in a few seconds[3] whereas $dom/ddeg$ allows to solve only 2 instances within 2 hours, and tested $mDVO$ only 6.

instances		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$	$mDVO$
$scen11$	cpu	16.28	3.64	93.97	21.05
(sat)	#asgs	2.69K	0.91K	31.81K	8.31K
$scen02$-$f25$	cpu	2.67	1.99	$> 2h$	292.27
(unsat)	#asgs	0.66K	0.89K	-	123.95K
$scen03$-$f10$	cpu	2.06	1.7	$> 2h$	1.80
(sat)	#asgs	0.68K	0.77K	-	0.42K
$scen03$-$f11$	cpu	4.63	2.93	$> 2h$	16.39
(unsat)	#asgs	0.75K	0.80K	-	1.09K
$scen06$-$w2$	cpu	0.31	0.93	$> 2h$	0.25
(unsat)	#asgs	0.02K	0.74K	-	0.01K
$scen07$-$w1$-$f4$	cpu	0.35	0.30	0.44	0.49
(sat)	#asgs	0.44K	0.48K	0.75K	0.76K
$scen07$-$w1$-$f5$	cpu	0.83	0.36	$> 2h$	$> 2h$
(unsat)	#asgs	0.44K	0.25K	-	-
$graph08$-$f10$	cpu	128.85	15.86	$> 2h$	$> 2h$
(sat)	#asgs	29.64K	5.77K	-	-
$graph08$-$f11$	cpu	2.32	17.71	$> 2h$	$> 2h$
(unsat)	#asgs	0.24K	5.07K	-	-
$graph14$-$f27$	cpu	185.61	5.25	$> 2h$	$> 2h$
(sat)	#asgs	39.84K	1.62K	-	-
$graph14$-$f28$	cpu	30.00	27.79	$> 2h$	$> 2h$
(unsat)	#asgs	4.60K	7.85K	-	-

Table 2. RLFAP instances

We have also experimented some realistic radar surveillance instances as proposed by the Swedish Institute of Computer Science (SICS). The problem is to adjust the signal strength (from 0 to 3) of a given number of fixed radars wrt 6 geographic sectors. Moreover, each cell of the geographic area must be covered exactly by 3 radar stations, except for some insignificant cells that must not be covered. We have artificially generated three sets of 50 instances involving non-binary constraints. Each set is denoted rs-i-j where i and j represent the number of radars and the number of insignificant cells, respectively. One can observe on Table 3 the huge gain obtained when using conflict-directed heuristics: after 2 hours (per instance) and $1,000$ to $10,000$ times as many assignments, classical heuristics do not succeed to solve all 50 instances.

instances		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$	$dom \oplus ddeg$
rs-30-0	cpu	0.05	0.09	623.04	$2,319.99$
	#asgs	0.20K	0.23K	$1,656.95$K	$12,538.74$K
(50/50 sat)	#solved	50	50	46	35
rs-24-12-ac	cpu	3.78	7.86	$2,020.06$	$2,336.01$
	#asgs	1.21K	1.68K	722.36K	641.92K
(0/50 sat)	#solved	50	50	36	35
rs-24-2	cpu	3.86	3.24	$2,030.73$	$2,545.44$
	#asgs	0.92K	0.59K	$4,027.56$K	$7,805.41$K
(26/50 sat)	#solved	50	50	37	33

Table 3. Radar surveillance instances

Next, we have dealt with the following academic instances:

- two chessboard coloration instances [2], denoted cc-7-2 and cc-7-3, involving quaternary constraints,
- two Golomb ruler instances[4], denoted gr-44-9 and gr-44-10, involving binary and ternary constraints,
- two prime queen attacking instances[5], denoted qa-5 and qa-6, involving only binary constraints.

Table 4 shows that conflict-directed heuristics are usually better than the other heuristics.

instances		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$	$mDVO$
cc-7-2	cpu	1.87	2.38	10.99	4.34
(unsat)	#asgs	2.23K	5.26K	30.19K	14.78K
cc-7-3	cpu	214.82	39.86	292.28	322.64
(sat)	#asgs	387.3K	104.38K	714.1K	780.56K
gr-44-9	cpu	31.57	60.54	37.00	47.91
(sat)	#asgs	5.39K	14.80K	11.74K	11.73K
gr-44-10	cpu	708.42	167.79	$1,356.64$	583.23
(unsat)	#asgs	75.56K	22.38K	202.07K	44.28K
qa-5	cpu	4.50	2.37	70.50	24.97
(sat)	#asgs	7.90K	4.79K	318.60K	100.81K
qa-6	cpu	88.73	82.46	5490.86	339.73
(sat)	#asgs	62.74K	74.58K	$7,703.44$K	523.14K

Table 4. Academic instances

Finally, to study the behaviour of the different heuristics wrt problems involving random generation, we have considered the following classes of instances :

- two classes of 100 satisfiable balanced Quasigroup With Holes (bQWH) instances [15] of order 15 with 106 holes and order 18 with 141 holes, respectively,
- three classes $< n, d, m, t >$ of 100 random instances situated at the phase transition where n denotes the number of variables, d the uniform domain size, m the number of binary constraints and t the constraint tightness.

In Table 5, we remark that for (structured or pure) random problems, as the constraint tightness grows, the gap between conflict-directed heuristics (and more particularly $dom/wdeg$) and the other ones becomes more and more important. However, this observation deserves further exploration.

[3] The (modified) RLFAP instances that have not been solved by MAC-$wdeg$ within 2 hours are built from $scen11$ by removing from 1 to 3 frequencies.

[4] See problem006 at http://4c.ucc.ie/~tw/csplib/

[5] See problem029 at http://4c.ucc.ie/~tw/csplib/

instances		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$	$dom \oplus ddeg$
$bqwh$	cpu	1.06	0.50	4.85	4.94
15-106	#asgs	1.9K	1.7K	21.2K	23.3K
(100/100 sat)	#solved	100	100	100	100
$bqwh$	cpu	20.12	8.90	705.07	719.16
18-141	#asgs	25.8K	17.7K	1,990.8K	2,143.3K
(100/100 sat)	#solved	100	100	98	98
$<80, 10,$	cpu	315.37	149.73	153.85	516.43
$400, 0.35>$	#asgs	373.4K	178.4K	196.4K	715.6K
(40/100 sat)	#solved	100	100	100	100
$<200, 10,$	cpu	79.45	52.39	96.72	2,317.23
$500, 0.55>$	#asgs	62.7K	44.0K	94.5K	2,810.1K
(14/100 sat)	#solved	100	100	100	51
$<900, 10,$	cpu	16.64	7.79	41.27	2,073.24
$1250, 0.70>$	#asgs	3.9K	2.8K	17.9K	1,107.3K
(43/100 sat)	#solved	100	100	100	44

Table 5. Structured and pure random instances

5 Related work

Dynamic weighting has been first introduced by [21] and [23] in order to improve the performance of local search methods. The Breakout method of [21] simply increases the weights of all current nogoods (prohibited tuples of constraints) whenever a local minimum is encountered. Such weights are then used to escape from local minima. Independently, [23] have proposed to increment the weight of all clauses (of a boolean formula in conjunctive normal form) not satisfied by the current assignment. This weighting strategy (combined with two other strategies: random walk and averaging-in) has been shown to dramatically enhance the applicability of a randomized greedy local search procedure (GSAT) for propositional satisfiability testing. In the context of applying local search to general CSPs, constraint weighting has been addressed by [26].

On the other hand, in [20], an hybrid search technique is proposed, combining a GSAT-like procedure with the well known DP procedure. The branching strategy of the logically-complete DP procedure is based on the dynamic constraint weighting managed by GSAT in order to direct search toward an inconsistent kernel. [6] have improved the satisfiability branching heuristics using clauses that shown previously unsatisfiable. Also, [7] used clause weighting to detect minimally unsatisfiable subformulae in SAT instances.

Finally, among specialized heuristics that have been proposed in the literature and that are related to our approach, one can cite the heuristics proposed by [22]. These heuristics, adapted to the job shop scheduling CSP, involve focusing search toward critical variables, i.e., variables that are most likely to be involved in a conflict.

6 Conclusion

In this paper, we have introduced a new generic variable ordering heuristic that learn from encountered failures to manage the choice of the variables to be assigned. From works of [21, 23, 20, 26, 7], we had imagined that locally inconsistent or hard parts of CSPs could be first examined by a systematic search algorithm using this conflict-directed heuristic. Experimentally, we have shown that our approach is the most efficient current one wrt significant and large classes of academic, random and real world instances.

As a perspective, we plan to compare the performance of our approach with other approaches that use sophisticated backjumping techniques (see, e.g. [9, 8, 18, 1]).

Acknowledgments

We would like to thank Carla Gomes for providing us software in order to reproduce bQWH instances. This paper has been supported by the CNRS, the "programme TACT de la Région Nord/Pas-de-Calais" and by the "IUT de Lens".

REFERENCES

[1] F. Bacchus, 'Extending forward checking', in *Proceedings of CP'00*, pp. 35–51, (2000).

[2] M. Beresin, E. Levin, and J. Winn, 'A chessboard coloring problem', *The College Mathematics Journal*, **20**(2), 106–114, (1989).

[3] C. Bessiere, A. Chmeiss, and L. Sais, 'Neighborhood-based variable ordering heuristics for the constraint satisfaction problem', in *Proceedings of CP'01*, pp. 565–569, (2001).

[4] C. Bessiere and J. Regin, 'MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems', in *Proceedings of CP'96*, pp. 61–75, (1996).

[5] D. Brelaz, 'New methods to color the vertices of a graph', *Communications of the ACM*, **22**, 251–256, (1979).

[6] L. Brisoux, E. Gregoire, and L. Sais, 'Improving backtrack search for sat by means of redundancy', in *Proceedings of ISMIS'99*, pp. 301–309, (1999).

[7] R. Bruni and A. Sassano, 'Detecting minimaly unsatisfiable subformulae in unsatisfiable SAT instances by means of adaptative core search', in *Proceedings of SAT'00*, (2000).

[8] X. Chen and P. van Beek, 'Conflict-directed backjumping revisited', *Journal of Artificial Intelligence Research*, **14**, 53–81, (2001).

[9] R. Dechter and D. Frost, 'Backjump-based backtracking for constraint satisfaction problems', *Artificial Intelligence*, **136**, 147–188, (2002).

[10] R. Dechter and I. Meiri, 'Experimental evaluation of preprocessing techniques in constraint satisfaction problems', in *Proceedings of IJCAI'89*, pp. 271–277, (1989).

[11] E.C. Freuder, 'A sufficient condition for backtrack-free search', *Journal of the ACM*, **29**(1), 24–32, (1982).

[12] D. Frost and R. Dechter, 'Look-ahead value ordering for constraint satisfaction problems', in *Proceedings of IJCAI'95*, pp. 572–578, (1995).

[13] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh, 'An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem', in *Proceedings of CP'96*, pp. 179–193, (1996).

[14] C.P. Gomes, B. Selman, N. Crato, and H. Kautz, 'Heavy-tailed phenomena in satisfiability and constraint satisfaction problems', *Journal of Automated Reasoning*, **24**, 67–100, (2000).

[15] C.P. Gomez and D. Shmoys, 'Completing quasigroups or latin squares: a structured graph coloring problem', in *Proceedings of Computational Symposium on Graph Coloring and Generalization*, (2002).

[16] R.M. Haralick and G.L. Elliott, 'Increasing tree search efficiency for constraint satisfaction problems', *Artificial Intelligence*, **14**, 263–313, (1980).

[17] M.C. Horsch and W.S. Havens, 'An empirical study of probabilistic arc consistency as a variable ordering heuristic', in *Proceedings of CP'00*, pp. 525–530, (2000).

[18] N. Jussien, R. Debruyne, and P. Boizumault, 'Maintaining arc-consistency within dynamic backtracking', in *Proceedings of CP'00*, pp. 249–261, (2000).

[19] C. Lecoutre, F. Boussemart, and F. Hemery, 'Exploiting multidirectionality in coarse-grained arc consistency algorithms', in *Proceedings of CP'03*, pp. 480–494, (2003).

[20] B. Mazure, L. Sais, and E. Gregoire, 'Boosting complete techniques thanks to local search methods', *Annals of Mathematics and Artificial Intelligence*, **22**, 319–331, (1998).

[21] P. Morris, 'The breakout method for escaping from local minima', in *Proceedings of AAAI'93*, pp. 40–45, (1993).

[22] N. Sadeh and M.S. Fox, 'Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem', *Artificial Intelligence*, **86**, 1–41, (1996).

[23] B. Selman and H. Kautz, 'Domain-independent extensions to gsat: solving large structured satisfiability problems', in *Proceedings of IJCAI'93*, pp. 290–295, (1993).

[24] B.M. Smith, 'The brelaz heuristic and optimal static orderings', in *Proceedings of CP'99*, pp. 405–418, Alexandria, VA, (1999).

[25] B.M. Smith and S.A. Grant, 'Trying harder to fail first', in *Proceedings of ECAI'98*, pp. 249–253, Brighton, UK, (1998).

[26] J.R. Thornton, *Constraint weighting local search for constraint satisfaction*, Ph.D. dissertation, Griffith University, Australia, 2000.

[27] R. Williams, C.P. Gomes, and B. Selman, 'Backdoors to typical case complexity', in *Proceedings of IJCAI'03*, (2003).