Nogood Recording from Restarts

Christophe Lecoutre and **Lakhdar Sais** and **Sébastien Tabary** and **Vincent Vidal**
CRIL−CNRS FRE 2499
Université d'Artois
Lens, France
$\{lecoutre, sais, tabary, vidal\}$@cril.univ-artois.fr

Abstract

In this paper, nogood recording is investigated within the randomization and restart framework. Our goal is to avoid the same situations to occur from one run to the next one. More precisely, nogoods are recorded when the current cutoff value is reached, i.e. before restarting the search algorithm. Such a set of nogoods is extracted from the last branch of the current search tree. Interestingly, the number of nogoods recorded before each new run is bounded by the length of the last branch of the search tree. As a consequence, the total number of recorded nogoods is polynomial in the number of restarts. Experiments over a wide range of CSP instances demonstrate the effectiveness of our approach.

1 Introduction

Nogood recording (or learning) has been suggested as a technique to enhance CSP (Constraint Satisfaction Problem) solving in [Dechter, 1990]. The principle is to record a nogood whenever a conflict occurs during a backtracking search. Such nogoods can then be exploited later to prevent the exploration of useless parts of the search tree. The first experimental results obtained with learning were given in the early 90's [Dechter, 1990; Frost and Dechter, 1994; Schiex and Verfaillie, 1994].

Contrary to CSP, the recent impressive progress in SAT (Boolean Satisfiability Problem) has been achieved using nogood recording (clause learning) under a randomization and restart policy enhanced with a very efficient lazy data structure [Moskewicz *et al.*, 2001]. Indeed, the interest of clause learning has arisen with the availability of large instances (encoding practical applications) which contain some structure and exhibit heavy-tailed phenomenon. Learning in SAT is a typical successful technique obtained from the cross fertilization between CSP and SAT: nogood recording [Dechter, 1990] and conflict directed backjumping [Prosser, 1993] have been introduced for CSP and later imported into SAT solvers [Bayardo and Shrag, 1997; Marques-Silva and Sakallah, 1996].

Recently, a generalization of nogoods, as well as an elegant learning method, have been proposed in [Katsirelos and Bacchus, 2003; 2005] for CSP. While standard nogoods correspond to variable assignments, generalized nogoods also involve value refutations. These generalized nogoods benefit from nice features. For example, they can compactly capture large sets of standard nogoods and are proved to be more powerful than standard ones to prune the search space.

As the set of nogoods that can be recorded might be of exponential size, one needs to achieve some restrictions. For example, in SAT, learned nogoods are not minimal and are limited in number using the First Unique Implication Point (First UIP) concept. Different variants have been proposed (e.g. relevance bounded learning [Bayardo and Shrag, 1997]), all of them attempt to find the best trade-off between the overhead of learning and performance improvements. Consequently, the recorded nogoods cannot lead to a complete elimination of redundancy in search trees.

In this paper, nogood recording is investigated within the randomization and restart framework. The principle of our approach is to learn nogoods from the last branch of the search tree before a restart, discarding already explored parts of the search tree in subsequent runs. Roughly speaking, we manage nogoods by introducing a global constraint with a dedicated filtering algorithm which exploits watched literals [Moskewicz *et al.*, 2001]. The worst-case time complexity of this propagation algorithm is O($n^2\gamma$) where n is the number of variables and γ the number of recorded nogoods. Besides, we know that γ is at most $nd\rho$ where d is the greatest domain size and ρ is the number of restarts already performed. Remark that a related approach has been proposed in [Baptista *et al.*, 2001] for SAT in order to obtain a complete restart strategy while reducing the number of recorded nogoods.

2 Technical Background

A Constraint Network (CN) P is a pair $(\mathscr{X}, \mathscr{C})$ where $\mathscr{X}$ is a set of n variables and $\mathscr{C}$ a set of e constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted $dom(X)$, which contains the set of values allowed for X. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, denoted $vars(C)$, and has an associated relation, denoted $rel(C)$, which contains the set of tuples allowed for $vars(C)$.

A solution to a CN is an assignment of values to all the variables such that all the constraints are satisfied. A CN is said to be satisfiable iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given CN is satisfiable. A CSP instance is then defined by a CN, and solving it involves either finding one (or more) solution or determining its unsatisfiability. To solve a CSP instance, one can modify the CN by using inference or search methods [Dechter, 2003].

The backtracking algorithm (BT) is a central algorithm for solving CSP instances. It performs a depth-first search in order to instantiate variables and a backtrack mechanism when dead-ends occur. Many works have been devoted to improve its forward and backward phases by introducing look-ahead and look-back schemes [Dechter, 2003]. Today, MAC [Sabin and Freuder, 1994] is the (look-ahead) algorithm considered as the most efficient generic approach to solve CSP instances. It maintains a property called Arc Consistency (AC) during search. When mentioning MAC, it is important to indicate which branching scheme is employed. Indeed, it is possible to consider binary (2-way) branching or non binary (d-way) branching. These two schemes are not equivalent as it has been shown that binary branching is more powerful (to refute unsatisfiable instances) than non-binary branching [Hwang and Mitchell, 2005]. With binary branching, at each step of search, a pair (X,v) is selected where X is an unassigned variable and v a value in dom(X), and two cases are considered: the assignment $X = v$ and the refutation $X \neq v$. The MAC algorithm (using binary branching) can then be seen as building a binary tree. Classically, MAC always starts by assigning variables before refuting values. Generalized Arc Consistency (GAC) (e.g. [Bessiere *et al.*, 2005]) extends AC to non binary constraints, and MGAC is the search algorithm that maintains GAC.

Although sophisticated look-back algorithms such as CBJ (Conflict Directed Backjumping) [Prosser, 1993] and DBT (Dynamic Backtracking) [Ginsberg, 1993] exist, it has been shown [Bessiere and Régin, 1996; Boussemart *et al.*, 2004; Lecoutre *et al.*, 2004] that MAC combined with a good variable ordering heuristic often outperforms such techniques.

3 Reduced nld-Nogoods

From now on, we will consider a search tree built by a backtracking search algorithm based on the 2-way branching scheme (e.g. MAC), positive decisions being performed first. Each branch of the search tree can then be seen as a sequence of positive and negative decisions, defined as follows:

Definition 1 *Let $P = (\mathscr{X},\mathscr{C})$ be a CN and (X,v) be a pair such that $X \in \mathscr{X}$ and $v \in dom(X)$. The assignment $X = v$ is called a positive decision whereas the refutation $X \neq v$ is called a negative decision. $\neg(X = v)$ denotes $X \neq v$ and $\neg(X \neq v)$ denotes $X = v$.*

Definition 2 *Let $\Sigma = \langle \delta_1, \ldots, \delta_i, \ldots, \delta_m \rangle$ be a sequence of decisions where δ_i is a negative decision. The sequence $\langle \delta_1, \ldots, \delta_i \rangle$ is called a nld-subsequence (negative last decision subsequence) of Σ. The set of positive and negative decisions of Σ are denoted by $pos(\Sigma)$ and $neg(\Sigma)$, respectively.*

Definition 3 *Let P be a CN and Δ be a set of decisions. $P|_\Delta$ is the CN obtained from P s.t., for any positive decision $(X = v) \in \Delta$, dom(X) is restricted to $\{v\}$, and, for any negative decision $(X \neq v) \in \Delta$, v is removed from dom(X).*

Definition 4 *Let P be a CN and Δ be a set of decisions. Δ is a nogood of P iff $P|_\Delta$ is unsatisfiable.*

From any branch of the search tree, a nogood can be extracted from each negative decision. This is stated by the following proposition:

Proposition 1 *Let P be a CN and Σ be the sequence of decisions taken along a branch of the search tree. For any nld-subsequence $\langle \delta_1, \ldots, \delta_i \rangle$ of Σ, the set $\Delta = \{\delta_1, \ldots, \neg\delta_i\}$ is a nogood of P (called nld-nogood)*[1].

Proof. As positive decisions are taken first, when the negative decision δ_i is encountered, the subtree corresponding to the opposite decision $\neg\delta_i$ has been refuted. □

These nogoods contain both positive and negative decisions and then correspond to the definition of generalized nogoods [Focacci and Milano, 2001; Katsirelos and Bacchus, 2005]. In the following, we show that nld-nogoods can be reduced in size by considering positive decisions only. Hence, we benefit from both an improvement in space complexity and a more powerful pruning capability.

By construction, CSP nogoods do not contain two opposite decisions i.e. both $x = v$ and $x \neq v$. Propositional resolution allows to deduce the clause $r = (\alpha \vee \beta)$ from the clauses $x \vee \alpha$ and $\neg x \vee \beta$. Nogoods can be represented as propositional clauses (disjunction of literals), where literals correspond to positive and negative decisions. Consequently, we can extend resolution to deal directly with CSP nogoods (e.g. [Mitchell, 2003]), called Constraint Resolution (C-Res for short). It can be defined as follows:

Definition 5 *Let P be a CN, and $\Delta_1 = \Gamma \cup \{x_i = v_i\}$ and $\Delta_2 = \Lambda \cup \{x_i \neq v_i\}$ be two nogoods of P. We define Constraint Resolution as C-Res$(\Delta_1, \Delta_2) = \Gamma \cup \Lambda$.*

It is immediate that C-Res(Δ_1, Δ_2) is a nogood of P.

Proposition 2 *Let P be a CN and Σ be the sequence of decisions taken along a branch of the search tree. For any nld-subsequence $\Sigma' = \langle \delta_1, \ldots, \delta_i \rangle$ of Σ, the set $\Delta = pos(\Sigma') \cup \{\neg\delta_i\}$ is a nogood of P (called reduced nld-nogood).*

Proof. We suppose that Σ contains $k \geq 1$ negative decisions, denoted by $\delta_{g_1}, \ldots, \delta_{g_k}$, in the order that they appear in Σ. The nld-subsequence of Σ with k negative decisions is $\Sigma'_1 = \langle \delta_1, \ldots, \delta_{g_1}, \ldots, \delta_{g_k} \rangle$. Its corresponding nld-nogood is $\Delta_1 = \{\delta_1, \ldots, \delta_{g_1}, \ldots, \delta_{g_{k-1}}, \ldots, \neg\delta_{g_k}\}$, $\delta_{g_{k-1}}$ being now the last negative decision. The nld-subsequence of Σ with $k-1$ negative decisions is $\Sigma'_2 = \langle \delta_1, \ldots, \delta_{g_1}, \ldots, \delta_{g_{k-1}} \rangle$. Its corresponding nld-nogood is $\Delta_2 = \{\delta_1, \ldots, \delta_{g_1}, \ldots, \neg\delta_{g_{k-1}}\}$. We now apply C-Res between Δ_1 and Δ_2 and we obtain $\Delta'_1 =$ C-Res$(\Delta_1, \Delta_2) = \{\delta_1, \ldots, \delta_{g_1}, \ldots, \delta_{g_{k-2}}, \ldots, \delta_{g_{k-1}-1}, \delta_{g_{k-1}+1}, \ldots, \neg\delta_{g_k}\}$. The last negative decision is now $\delta_{g_{k-2}}$, which will be eliminated with the nld-nogood containing $k-2$ negative decisions. All the remaining negative decisions are then eliminated by applying the same process. □

One interesting aspect is that the space required to store all nogoods corresponding to any branch of the search tree is

[1]The notation $\{\delta_1, \ldots, \neg\delta_i\}$ corresponds to $\{\delta_j \in \Sigma \mid j < i\} \cup \{\neg\delta_i\}$, reduced to $\{\neg\delta_1\}$ when $i = 1$.

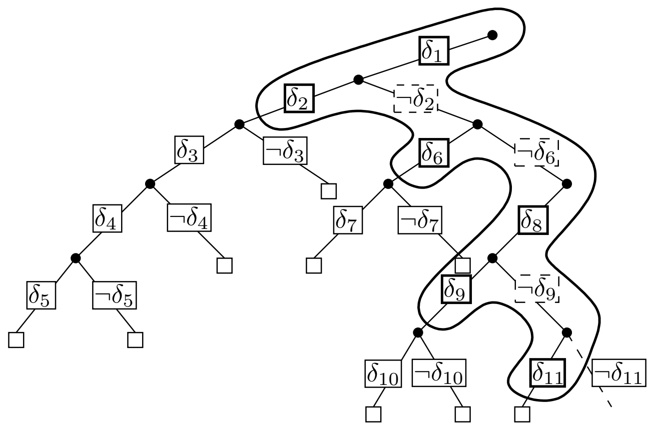

Figure 1: Area of nld-nogoods in a partial search tree

polynomial with respect to the number of variables and the greatest domain size.

Proposition 3 *Let P be a CN and Σ be the sequence of decisions taken along a branch of the search tree. The space complexity to record all nld-nogoods of Σ is O(n^2d^2) while the space complexity to record all reduced nld-nogoods of Σ is O(n^2d).*

Proof. First, the number of negative decisions in any branch is O(nd). For each negative decision, we can extract a (reduced) nld-nogood. As the size of any (resp. reduced) nld-nogood is O(nd) (resp. O(n) since it only contains positive decisions), we obtain an overall space complexity of O(n^2d^2) (resp. O(n^2d)). □

4 Nogood Recording from Restarts

In [Gomes *et al.*, 2000], it has been shown that the runtime distribution produced by a randomized search algorithm is sometimes characterized by an extremely long tail with some infinite moment. For some instances, this heavy-tailed phenomenon can be avoided by using random restarts, i.e. by restarting search several times while randomizing the employed search heuristic. For constraint satisfaction, restarts have been shown productive. However, when learning is not exploited (as it is currently the case for most of the academic and commercial solvers), the average performance of the solver is damaged (cf. Section 6).

Nogood recording has not yet been shown to be quite convincing for CSP (one noticeable exception is [Katsirelos and Bacchus, 2005]) and, further, it is a technique that leads, when uncontrolled, to an exponential space complexity. We propose to address this issue by combining nogood recording and restarts in the following way: reduced nld-nogoods are recorded from the last (and current) branch of the search tree between each run. Our aim is to benefit from both restarts and learning capabilities without sacrificing solver performance and space complexity.

Figure 1 depicts the partial search tree explored when the solver is about to restart. Positive decisions being taken first, a δ_i (resp. $\neg\delta_i$) corresponds to a positive (resp. negative) decision. Search has been stopped after refuting δ_{11} and taking the decision $\neg\delta_{11}$. The nld-nogoods of P are the following: $\Delta_1 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \delta_{11}\}$, $\Delta_2 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \delta_9\}$, $\Delta_3 = \{\delta_1, \neg\delta_2, \delta_6\}$, $\Delta_4 = \{\delta_1, \delta_2\}$. The first reduced nld-nogood is obtained as follows:

$$\begin{aligned}\Delta_1' &= \text{C-Res}(\text{C-Res}(\text{C-Res}(\Delta_1, \Delta_2), \Delta_3), \Delta_4)\\ &= \text{C-Res}(\text{C-Res}(\{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \delta_{11}\}, \Delta_3), \Delta_4)\\ &= \text{C-Res}(\{\delta_1, \neg\delta_2, \delta_8, \delta_{11}\}, \Delta_4)\\ &= \{\delta_1, \delta_8, \delta_{11}\}\end{aligned}$$

Applying the same process to the other nld-nogoods, we obtain:

$$\begin{aligned}\Delta_2' &= \text{C-Res}(\text{C-Res}(\Delta_2, \Delta_3), \Delta_4) = \{\delta_1, \delta_8, \delta_9\}.\\ \Delta_3' &= \text{C-Res}(\Delta_3, \Delta_4) = \{\delta_1, \delta_6\}.\\ \Delta_4' &= \Delta_4 = \{\delta_1, \delta_2\}.\end{aligned}$$

In order to avoid exploring the same parts of the search space during subsequent runs, recorded nogoods can be exploited. Indeed, it suffices to control that the decisions of the current branch do not contain all decisions of one nogood. Moreover, the negation of the last unperformed decision of any nogood can be inferred as described in the next section. For example, whenever the decision δ_1 is taken, we can infer $\neg\delta_2$ from nogood Δ_4' and $\neg\delta_6$ from nogood Δ_3'.

Finally, we want to emphasize that *reduced nld-nogoods extracted from the last branch subsume all reduced nld-nogoods that could be extracted from any branch previously explored*.

5 Managing Nogoods

In this section, we now show how to efficiently exploit reduced nld-nogoods by combining watched literals with propagation. We then obtain an efficient propagation algorithm enforcing GAC on all learned nogoods that can be collectively considered as a global constraint.

5.1 Recording Nogoods

Nogoods derived from the current branch of the search tree (i.e. reduced nld-nogoods) when the current run is stopped can be recorded by calling the $storeNogoods$ function (see Algorithm 1). The parameter of this function is the sequence of literals labelling the current branch. As observed in Section 3, a reduced nld-nogood can be recorded from each negative decision occurring in this sequence. From the root to the leaf of the current branch, when a positive decision is encountered, it is recorded in the set Δ (line 4), and when a negative decision is encountered, we record a nogood from the negation of this decision and all recorded positive ones (line 9). If the nogood is of size 1 (i.e. $\Delta = \emptyset$), it can be directly exploited by reducing the domain of the involved variable (see line 7). Otherwise, it is recorded, by calling the $addNogood$ function (not described here), in a structure exploiting watched literals [Moskewicz *et al.*, 2001].

We can show that the worst-case time complexity of $storeNogoods$ is O($\lambda_p\lambda_n$) where λ_p and λ_n are the number of positive and negative decisions on the current branch, respectively.

Algorithm 1 storeNogoods($\Sigma = \langle \delta_1, \ldots, \delta_m \rangle$)

$\Delta \leftarrow \emptyset$
for $i \in [1, m]$ **do**
  if δ_i is a positive decision **then**
    $\Delta \leftarrow \Delta \cup \{\delta_i\}$
  else
    if $\Delta = \emptyset$ **then**
      with $\delta_i = (X, v)$, remove v from dom(X)
    else
      addNogood($\Delta \cup \{\neg\delta_i\}$)
    end if
  end if
end for

Algorithm 2 propagate(S : Set of variables) : Boolean

1: $Q \leftarrow S$
2: **while** $Q \neq \emptyset$ **do**
3:   pick and delete X from Q
4:   **if** $\mid dom(X) \mid = 1$ **then**
5:     let v be the unique value in dom(X)
6:     **if** checkWatches($X = v$) = $false$ **then** return false
7:   **end if**
8:   **for** each $C \mid X \in vars(C)$ **do**
9:     **for** each $Y \in Vars(C) \mid X \neq Y$ **do**
10:       **if** revise(C,Y) **then**
11:         **if** $dom(Y) = \emptyset$ **then** return false
12:         **else** $Q \leftarrow Q \cup \{Y\}$
13: **end while**
14: return true

5.2 Exploiting Nogoods

Inferences can be performed using nogoods while establishing (maintaining) Generalized Arc Consistency. We show it with a coarse-grained GAC algorithm based on a variable-oriented propagation scheme [McGregor, 1979]. The Algorithm 2 can be applied to any CN (involving constraints of any arity) in order to establish GAC. At preprocessing, $propagate$ must be called with the set S of variables of the network whereas during search, S only contains the variable involved in the last positive or negative decision. At any time, the principle is to have in Q all variables whose domains have been reduced by propagation.

Initially, Q contains all variables of the given set S (line 1). Then, iteratively, each variable X of Q is selected (line 3). If dom(X) corresponds to a singleton $\{v\}$ (lines 4 to 7), we can exploit recorded nogoods by checking the consistency of the nogood base. This is performed by the function $checkWatches$ (not described here) by iterating all nogoods involving $X = v$ as watched literal. For each such nogood, either another literal not yet watched can be found, or an inference is performed (and the set Q is updated).

The rest of the algorithm (lines 8 to 12) corresponds to the body of a classical generic coarse-grained GAC algorithm. For each constraint C binding X, we perform the revision of all arcs (C, Y) with $Y \neq X$. A revision is performed by a call to the function $revise$, specific to the chosen coarse-grained arc consistency algorithm, and entails removing values that became inconsistent with respect to C. When the revision of an arc (C, Y) involves the removal of some values in $dom(Y)$, $revise$ returns $true$ and the variable Y is added to Q. The algorithm loops until a fixed point is reached.

The worst-case time complexity of $checkWatches$ is O($n\gamma$) where γ is the number of reduced nld-nogoods stored in the base and n is the number of variables[2]. Indeed, in the worst case, each nogood is watched by the literal given in parameter, and the time complexity of dealing with a reduced nld-nogood in order to find another watched literal or make an inference is O(n). Then, the worst-case time complexity of $propagate$ is O($er^2d^r + n^2\gamma$) where r is the greatest constraint arity. More precisely, the cost of establishing GAC (using a generic approach) is O(er^2d^r) when an algorithm such as GAC2001 [Bessiere *et al.*, 2005] is used and the cost of exploiting nogoods to enforce GAC is O($n^2\gamma$). Indeed, $checkWatches$ is O($n\gamma$) and it can be called only once per variable.

The space complexity of the structures introduced to manage reduced nld-nogoods in a backtracking search algorithm is O($n(d + \gamma)$). Indeed, we need to store γ nogoods of size at most n and we need to store watched literals which is O(nd).

[2] In practice, the size of reduced nld-nogoods can be far smaller than n (cf. Section 6).

6 Experiments

In order to show the practical interest of the approach described in this paper, we have conducted an extensive experimentation (on a PC Pentium IV 2.4GHz 512Mo under Linux). We have used the Abscon solver to run M(G)AC2001 (denoted by MAC) and studied the impact of exploiting restarts (denoted by MAC+RST) and nogood recording from restarts (denoted by MAC+RST+NG). Concerning the restart policy, the initial number of allowed backtracks for the first run has been set to 10 and the increasing factor to 1.5 (i.e., at each new run, the number of allowed backtracks increases by a 1.5 factor). We used three different variable ordering heuristics: the classical $brelaz$ [Brelaz, 1979] and $dom/ddeg$ [Bessiere and Régin, 1996] as well as the adaptive $dom/wdeg$ that has been recently shown to be the most efficient generic heuristic [Boussemart *et al.*, 2004; Lecoutre *et al.*, 2004; Hulubei and O'Sullivan, 2005; van Dongen, 2005]. Importantly, when restarts are performed, randomization is introduced in $brelaz$ and $dom/ddeg$ to break ties. For $dom/wdeg$, the weight of constraints are preserved from each run to the next one, which makes randomization useless (weights are sufficiently discriminant).

In our first experimentation, we have tested the three algorithms on the full set of 1064 instances used as benchmarks for the first competition of CSP solvers [van Dongen, 2005]. The time limit to solve an instance was fixed to 30 minutes. Table 1 provides an overview of the results in terms of the number of instances unsolved within the time limit ($\#timeouts$) and the average cpu time in seconds (avg $time$) computed from instances solved by all three methods. Figures 2 and 3 represent scatter plots displaying pairwise comparisons for $dom/ddeg$ and $dom/wdeg$. Finally, Table 2 focuses on the most difficult real-world instances of the Radio Link Frequency Assignment Problem (RLFAP). Performance is measured in terms of the cpu time in seconds (no timeout)

		MAC		
			$+RST$	$+RST+NG$
$dom/ddeg$	$\#timeouts$	365	378	337
	$avg\ time$	125.0	159.0	109.1
$brelaz$	$\#timeouts$	277	298	261
	$avg\ time$	85.1	121.7	78.2
$dom/wdeg$	$\#timeouts$	140	123	121
	$avg\ time$	47.8	56.0	43.6

Table 1: Number of unsolved instances and average cpu time on the 2005 CSP competition benchmarks, given 30 minutes CPU.

and the number of visited nodes. An analysis of all these results reveals three main points.

Restarts (without learning) yields mitigated results. First, we observe an increased average cpu time for all heuristics and fewer solved instances for classical ones. However, a close look at the different series reveals that MAC+RST combined with $brelaz$ (resp. $dom/ddeg$) solved 27 (resp. 32) less instances than MAC on the series ehi. These instances correspond to random instances embedding a small unsatisfiable kernel. As classical heuristics do not guide search towards this kernel, restarting search tends to be nothing but an expense. Without these series, MAC+RST would have solved more instances than MAC (but, still, with worse performance). Also, remark that $dom/wdeg$ renders MAC+RST more robust than MAC (even on the ehi series).

Nogood recording from restarts improves MAC performance. Indeed, both the number of unsolved instances and the average cpu time are reduced. This is due to the fact that the solver never explores several times the same portion of the search space while benefiting from restarts.

Nogood recording from restarts applied to real-world instances pays off. When focusing to the hardest instances [van Dongen, 2005] built from the real-world RLFAP instance $scen$-11, we can observe in Table 2 that using a restart policy allows to be more efficient by almost one order of magnitude. When we further exploit nogood recording, the gain is about 10%.

Finally, we noticed that the number and the size of the reduced nld-nogoods recorded during search were always very limited. As an illustration, let us consider the hardest RLFAP instance $scen11-f1$ which involves 680 variables and a greatest domain size of 43 values. MAC+RST+NG solved this instance in 36 runs while only 712 nogoods of average size 8.5 and maximum size 33 were recorded.

7 Conclusion

In this paper, we have studied the interest of recording nogoods in conjunction with a restart strategy. The benefit of restarting search is that the heavy-tailed phenomenon observed on some instances can be avoided. The drawback is that we can explore several times the same parts of the search tree. We have shown that it is quite easy to eliminate this drawback by recording a set of nogoods at the end of each run. For efficiency reasons, nogoods are recorded in a base (and so do not correspond to new constraints) and propagation is performed using the 2-literal watching technique introduced for SAT. One can consider the base of nogoods as a unique global constraint with an efficient associated propagation algorithm.

		MAC		
			$+RST$	$+RST+NG$
scen11-f12	cpu	0.85	0.84	0.84
	$nodes$	695	477	445
scen11-f10	cpu	0.95	0.82	1.03
	$nodes$	862	452	636
scen11-f8	cpu	14.6	1.8	1.9
	$nodes$	14068	1359	1401
scen11-f7	cpu	185	9.4	8.4
	$nodes$	$207K$	9530	8096
scen11-f6	cpu	260	21.8	16.9
	$nodes$	$302K$	22002	16423
scen11-f5	cpu	1067	105	82.3
	$nodes$	$1327K$	$117K$	90491
scen11-f4	cpu	2494	367	339
	$nodes$	$2826K$	$419K$	$415K$
scen11-f3	cpu	9498	1207	1035
	$nodes$	$12M$	$1517K$	$1286K$
scen11-f2	cpu	$29K$	3964	3378
	$nodes$	$37M$	$5011K$	$4087K$
scen11-f1	cpu	$69K$	9212	8475
	$nodes$	$93M$	$12M$	$10M$

Table 2: Performance on hard RLFAP Instances using the $dom/wdeg$ heuristic (no timeout)

Our experimental results show the effectiveness of our approach since the state-of-the-art generic algorithm MAC-dom/wdeg is improved. Our approach not only allows to solve more instances than the classical approach within a given timeout, but also is, on the average, faster on instances solved by both approaches.

Acknowledgments

This paper has been supported by the CNRS and the ANR "Planevo" project n^{o}JC05_41940.

References

[Baptista *et al.*, 2001] L. Baptista, I. Lynce, and J. Marques-Silva. Complete search restart strategies for satisfiability. In *Proceedings of SSA'01 workshop held with IJCAI'01*, 2001.

[Bayardo and Shrag, 1997] R.J. Bayardo and R.C. Shrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of AAAI'97*, pages 203–208, 1997.

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

[Bessiere *et al.*, 2005] C. Bessiere, J.C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[Boussemart *et al.*, 2004] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

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

[Dechter, 1990] R. Dechter. Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[Dechter, 2003] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

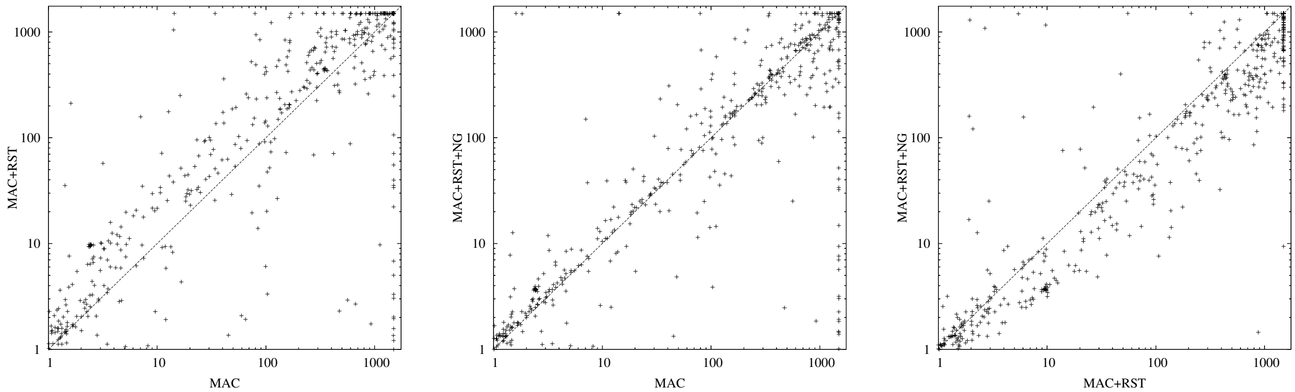

Figure 2: Pairwise comparison (cpu time) on the 2005 CSP competition benchmarks using the dom/ddeg heuristic

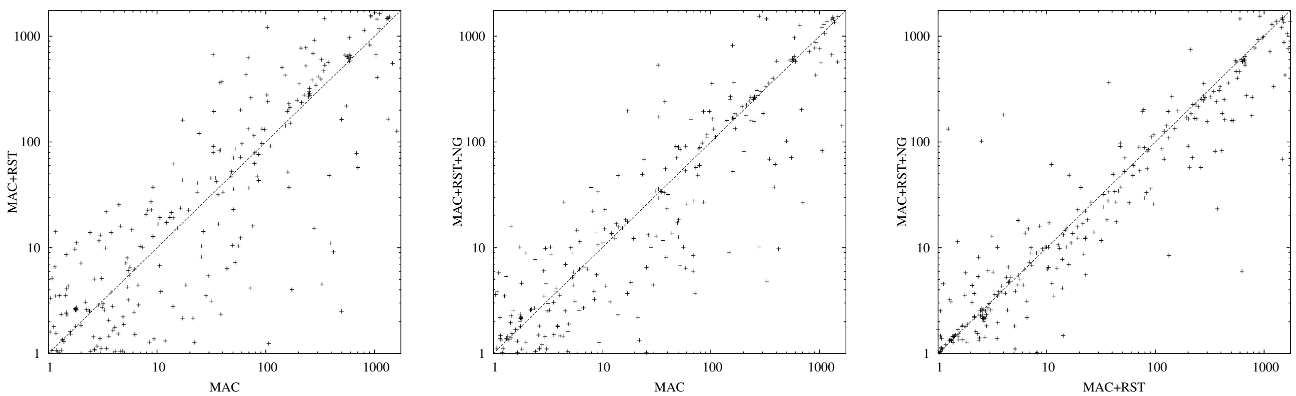

Figure 3: Pairwise comparison (cpu time) on the 2005 CSP competition benchmarks using the dom/wdeg heuristic

[Focacci and Milano, 2001] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP'01*, pages 77–92, 2001.

[Frost and Dechter, 1994] D. Frost and R. Dechter. Dead-end driven learning. In *Proc. of AAAI'94*, pages 294–300, 1994.

[Ginsberg, 1993] M. Ginsberg. Dynamic backtracking. *Artificial Intelligence*, 1:25–46, 1993.

[Gomes *et al.*, 2000] 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.

[Hulubei and O'Sullivan, 2005] T. Hulubei and B. O'Sullivan. Search heuristics and heavy-tailed behaviour. In *Proceedings of CP'05*, pages 328–342, 2005.

[Hwang and Mitchell, 2005] J. Hwang and D.G. Mitchell. 2-way vs d-way branching for CSP. In *Proceedings of CP'05*, pages 343–357, 2005.

[Katsirelos and Bacchus, 2003] G. Katsirelos and F. Bacchus. Unrestricted nogood recording in CSP search. In *Proceedings of CP'03*, pages 873–877, 2003.

[Katsirelos and Bacchus, 2005] G. Katsirelos and F. Bacchus. Generalized nogoods in CSPs. In *Proceedings of AAAI'05*, pages 390–396, 2005.

[Lecoutre *et al.*, 2004] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI'04*, pages 549–557, 2004.

[Marques-Silva and Sakallah, 1996] J.P. Marques-Silva and K.A. Sakallah. Conflict analysis in search algorithms for propositional satisfiability. RT/4/96, INESC, Lisboa, Portugal, 1996.

[McGregor, 1979] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.

[Mitchell, 2003] D.G. Mitchell. Resolution and constraint satisfaction. In *Proceedings of CP'03*, pages 555–569, 2003.

[Moskewicz *et al.*, 2001] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of DAC'01*, pages 530–535, 2001.

[Prosser, 1993] P. Prosser. Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

[Sabin and Freuder, 1994] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.

[Schiex and Verfaillie, 1994] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *IJAIT*, 3(2):187–207, 1994.

[van Dongen, 2005] M.R.C. van Dongen, editor. *Proceedings of CPAI'05 workshop held with CP'05*, volume II, 2005.