Combining Nogoods in Restart-Based Search

Gael Glorian, Frederic Boussemart, Jean-Marie Lagniez, Christophe Lecoutre, and Bertrand Mazure

CRIL, Univ. Artois and CNRS, F62300 Lens, France
{glorian, boussemart, lagniez, lecoutre, mazure}@cril.fr

Abstract

Nogood recording is a form of learning that has been shown useful for solving constraint satisfaction problems. One simple approach involves recording nogoods that are extracted from the rightmost branches of the successive trees built by a backtrack search algorithm with restarts. In this paper, we propose several mechanisms to reason with so-called increasing-nogoods that exactly correspond to the states reached at the end of each search run. Interestingly, some similarities that can be observed between increasing-nogoods allow us to propose new original ways of dynamically combining them in order to improve the overall filtering capability of the learning system. Our preliminary results show the practical interest of our approach.

1 Introduction

Nogood recording is a learning technique that has been applied to the Constraint Satisfaction Problem (CSP) in the 90's [2, 4, 14]. A classical nogood is defined as a partial instantiation that cannot be extended into a solution. Such nogoods have been cleverly exploited to manage explanations [5, 7] of values that are deleted during search (when running constraint propagation). They have also been generalized [8] by incorporating both assigned variables (positive decisions) and refuted values (negative decisions). More recently, the practical interest of nogood recording has been revisited in the context of lazy clause generation [3].

Nogoods can also be effective in the context of a backtrack search algorithm that regularly triggers restarts. Indeed, just before restarting, a set of nogoods can be easily identified [10] on the rightmost branch of the search tree, which stands for the part of search space that has been explored during the last run. By recording these so-called nld-nogoods, we obtain the guarantee of never exploring the same subtrees, further making the approach complete. This restart-based learning mechanism has been extended to take into account symmetry breaking [11, 13] and the increasing nature of nld-nogoods [12], called increasing-nogoods for this reason.

In this paper, we propose several mechanisms to combine increasing-nogoods, allowing us to increase their filtering capacity. By dynamically analyzing relevant subsets of increasing-nogoods, especially from equivalence forms between decisions, we show that the search space can be more efficiently pruned. More specifically, we introduce three inference rules for deeper reasoning with increasing-nogoods.

2 Preliminaries

A constraint network P is a pair $(\mathcal{X}, \mathcal{C})$, where $\mathcal{X}$ is a finite set of variables and $\mathcal{C}$ a finite set of constraints. Each variable $x \in \mathcal{X}$ has a domain, denoted by $dom(x)$, which is the finite set of values a that can be assigned to x. Each constraint $c \in \mathcal{C}$ involves an ordered set of variables, called the scope of c and denoted by $scp(c)$. A constraint c is semantically defined by a relation, denoted by $rel(c)$, which is the set of tuples allowed by (variables of) c. Let $X \subseteq \mathcal{X}$ be a subset of variables, an instantiation I of

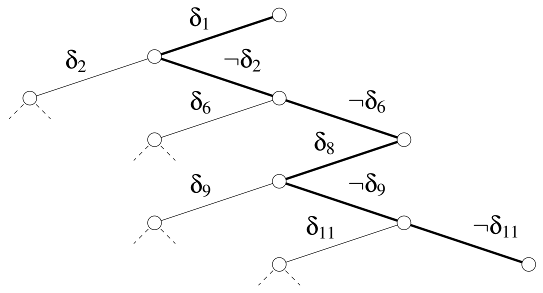

Figure 1: The search tree at the end of a run.

X maps each variable $x \in X$ to a value in $dom(x)$; we note $I[x] = a$ and $vars(I) = X$. An instantiation I is *complete* iff $vars(I) = X$, *partial* otherwise. A solution of P is a complete instantiation satisfying all constraints of P.

A nogood is an instantiation that cannot be extended to any solution. The benefit of recording nogoods is to avoid some form of thrashing, *i.e.* exploring the same unsatisfiable subtrees several times. There are two classical methods to identify and store nogoods: during search or at restarts. In this paper, we consider a complete backtrack search algorithm with binary branching and nogood recording from restarts [9]. Decisions taken during search are either positive (i.e., variable assignments such as $x = a$) or negative (i.e., value refutations such as $x \neq a$). A decision δ is checked to be positive or negative by simply writing $pos(\delta)$ and $neg(\delta)$, respectively. The variable involved in a decision δ is denoted by $var(\delta)$, whereas the value involved in a decision δ is denoted by $val(\delta)$. Binary branching means that at each search node a left branch labeled with a positive decision $x = a$ is developed first, and a right branch labeled with a negative decision $x \neq a$ is developed next. A decision δ is *satisfied* (resp., *falsified*) iff it holds (resp., does not hold) whatever is the value chosen in the current domain of $var(\delta)$. A decision that is not satisfied (resp., falsified) is said to be *unsatisfied* (resp., *unfalsified*).

3 Increasing Nogoods

So-called nld-nogoods [9] (negative last decision nogoods) can be extracted at each restart of a backtrack search algorithm. Let us assume that the sequence of labels all along the rightmost branch of a current search tree being developed is $\Sigma = \langle \delta_1, \ldots, \delta_m \rangle$, where each decision of Σ is either a positive or a negative decision. It is known that for any i such that $1 \leq i \leq m$ and $neg(\delta_i)$, the set $\{\delta_j : 1 \leq j < i \wedge pos(\delta_j)\} \cup \{\neg\delta_i\}$ is a reduced nld-nogood. Note that it only contains positive decisions (and so, is a standard nogood). From now on, for simplicity reasons, we simply call them nld-nogoods.

Example 3.1. *Let us consider the search tree depicted in Figure 1, where the rightmost branch of the tree is* $\Sigma = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \neg\delta_{11}\}$. *The following (reduced) nld-nogoods can be extracted:* $\{\delta_1, \delta_2\}$, $\{\delta_1, \delta_6\}$, $\{\delta_1, \delta_8, \delta_9\}$ *and* $\{\delta_1, \delta_8, \delta_{11}\}$.

As we can observe, there are some similarities between these nld-nogoods: they are said to be *increasing* [12, 13]. An increasing-nogood compactly represents the full set of nld-nogoods that can be extracted from a branch. To obtain an increasing-nogood from a set of nld-nogoods, we have first to consider each nld-nogood under its *directed* form.

Example 3.2. *Considering again the search tree in Figure 1, let us assume the decisions of the last branch represent:* $\Sigma = \langle x_2 = 1,\ x_3 \neq 0,\ x_4 \neq 1,\ x_5 = 2,\ x_1 \neq 1,\ x_6 \neq 2 \rangle$. *The four nld-nogoods* ng_0, ng_1,

ng_2 and ng_3 are given below under their logical forms (middle) and directed forms (right):

$$\begin{aligned}
ng_0 &\equiv \neg(& x_2 = 1 \wedge x_3 = 0) &\equiv & x_2 = 1 \Rightarrow x_3 \neq 0\\
ng_1 &\equiv \neg(& x_2 = 1 \wedge x_4 = 1) &\equiv & x_2 = 1 \Rightarrow x_4 \neq 1\\
ng_2 &\equiv \neg(& x_2 = 1 \wedge x_5 = 2 \wedge x_1 = 1) &\equiv & x_2 = 1 \wedge x_5 = 2 \Rightarrow x_1 \neq 1\\
ng_3 &\equiv \neg(& x_2 = 1 \wedge x_5 = 2 \wedge x_6 = 2) &\equiv & x_2 = 1 \wedge x_5 = 2 \Rightarrow x_6 \neq 2
\end{aligned}$$

In [12], the authors have shown that the set of directed nld-nogoods extracted from a branch are necessarily increasing, meaning that $\texttt{LHS}(ng_i) \subseteq \texttt{LHS}(ng_{i+1})$ where LHS designates the left hand side of the implication. This is illustrated on our example by:

$$\begin{aligned}
ng_0 &\equiv & x_2 = 1 \Rightarrow x_3 \neq 0\\
ng_1 &\equiv & LHS(ng_0) \Rightarrow x_4 \neq 1\\
ng_2 &\equiv & LHS(ng_1) \wedge x_5 = 2 \Rightarrow x_1 \neq 1\\
ng_3 &\equiv & LHS(ng_2) \Rightarrow x_6 \neq 2
\end{aligned}$$

In practical terms, it means that it suffices to record the branch exactly as it is instead of extracting nld-nogoods independently. Another important observation is that each increasing-nogood can be viewed as a constraint, together with a filtering algorithm enforcing GAC (Generalized Arc Consistency). Interestingly enough, if $\langle ng_1, \ldots, ng_t \rangle$ is a sequence of increasing nld-nogoods, and if $LHS(ng_i)$ contains two unsatisfied decisions then any nogood ng_j with $j \geq i$ is necessarily GAC because the LHS of larger nogoods subsume the LHS of smaller ones.

Technically, two indices α and β can be used to watch the two leftmost unsatisfied positive decisions in the sequence of an increasing-nogood. These two watched decisions as well as all the negative decisions that may occur between them are under surveillance, as δ_1, $\neg\delta_2$ and δ_3 in the following illustration:

$$\Sigma = \langle \overbrace{\underbrace{\delta_1}_{\alpha}, \neg\delta_2, \underbrace{\delta_3}_{\beta}}^{\text{Watched}}, \delta_4, \neg\delta_5, \neg\delta_6 \rangle$$

For the sake of simplicity, we consider that for any increasing-nogood Σ, the decisions in Σ that are watched by the alpha and beta indices can be respectively accessed by using $\alpha(\Sigma)$ and $\beta(\Sigma)$. On our illustration, $\alpha(\Sigma)$ and $\beta(\Sigma)$ are respectively δ_1 and δ_3.

The filtering algorithm (called IncNG) associated with an increasing-nogood (constraint) is triggered in three cases:

1. a watched negative decision is falsified: $\alpha(\Sigma)$ must be forced to be falsified, and consequently all nogoods within the constraint are satisfied;
2. $\alpha(\Sigma)$ is satisfied: all negative decisions between α and β must be satisfied and we search for the next unsatisfied positive decision;
3. $\beta(\Sigma)$ is satisfied: we need to find the next unsatisfied positive decision.

4 Reasoning with Increasing Nogoods

When considered as constraints, it is quite natural that nld-nogoods and increasing-nogoods are solicited independently for filtering tasks. However, we show that it is possible to exploit the similarities that exist (rather frequently) between such nogoods. More specifically, we introduce in this section three rules for reasoning deeper with increasing-nogoods.

Algorithm 1: checkNegativeDecisions(Σ : increasing-nogood)

```
1 foreach $x \in \texttt{diffVars}(\Sigma)$ do
2     if $dom(x) \subseteq \texttt{diffValues}(\Sigma, x)$ then
3         falsify $\alpha(\Sigma)$;
```

4.1 Reasoning with Watched Negative Decisions

By checking for each variable x and each increasing-nogood Σ that there exists a value in $dom(x)$ which is not involved in a negative decision for x between $\alpha(\Sigma)$ and $\beta(\Sigma)$, we have the guarantee of not missing some inferences from a simple reasoning on watched negative decisions.

Example 4.1. *Consider the following increasing-nogood:* $\Sigma = \langle x_2 = 1, x_3 \neq 2, x_3 \neq 4, x_5 = 3\rangle$. *Assume that we have* $\alpha(\Sigma)$ *and* $\beta(\Sigma)$ *being indices for* $x_2 = 1$ *and* $x_5 = 3$*, and all variables with the same domain* $\{1,2,3,4\}$*. If* x_2 *is assigned to the value* 1*, then the values* 2 *and* 4 *can be removed from* $dom(x_3)$*. Of course, a conflict occurs if* $dom(x_3)$ *only contains these two values. However, this conflict could have been avoided (anticipated) by removing the value* 1 *from* $dom(x_2)$ *as soon as* $dom(x_3)$ *is reduced to* $\{2,4\}$. $\square$

First, we introduce a function $\texttt{diffValues}(\Sigma, x_i)$ that returns for a given increasing-nogood Σ, the set of values present in a negative decision of Σ involving x_i and situated between $\alpha(\Sigma)$ and $\beta(\Sigma)$. We also introduce a function $\texttt{diffVars}(\Sigma)$ that returns the set of variables involved in a negative decision of Σ situated between $\alpha(\Sigma)$ and $\beta(\Sigma)$. For example, for $\Sigma = \langle x_2 = 1, x_3 \neq 2, x_3 \neq 4, x_5 = 3\rangle$ with $\alpha(\Sigma)$ being $x_2 = 1$ and $\beta(\Sigma)$ being $x_5 = 3$, we have $\texttt{diffVars}(\Sigma) = \{x_3\}$ and $\texttt{diffValues}(\Sigma, x_3) = \{2,4\}$. Algorithm 1 implements this way of reasoning, i.e. performs an inference by refuting the value involved in $\alpha(\Sigma)$, each time a conflict can be anticipated as discussed above. Even if increasing-nogoods are still reviewed independently (in turn), the filtering capability of the algorithm proposed in [12] is clearly improved if this simple procedure is systematically called. The worst-case time complexity of Algorithm 1 is $O(nd)$ where n is the number of variables and d is the size of the largest domain. Indeed, we can precompute sets $\texttt{diffVars}(\Sigma)$ and $\texttt{diffValues}(\Sigma, x)$ by scanning the decisions in Σ whose size is $O(nd)$. With these precomputed sets, executing lines 1–2 is also in $O(nd)$.

4.2 Combining Increasing Nogoods of Similar α

In this section, we extend the principle presented above to sets of increasing-nogoods. For this purpose, we partition the set of increasing-nogoods according to the decisions indexed by α: two increasing-nogoods Σ_i and Σ_j are in the same group iff $\alpha(\Sigma_i)$ is the same decision as $\alpha(\Sigma_j)$. Of course, it is therefore necessary to update the partition each time one α is modified (i.e., when filtering and backtracking). Despite that, reasoning about groups of increasing-nogoods allows us to improve the filtering capacity of the increasing-nogoods, and turns out to encompass the previous case.

Example 4.2. *Let us consider the three following increasing-nogoods:*

$$\Sigma_0 \equiv \quad \ldots,\ x_6 \neq 2,\ \underbrace{\underline{x_2 = 1}}_{\alpha},\ x_1 \neq 3,\ \underline{x_3 \neq 1},\ \ldots$$

$$\Sigma_1 \equiv \quad \ldots,x_2 \neq 0,\ x_1 \neq 2,\ \underbrace{\underline{x_2 = 1}}_{\alpha},\ \underline{x_3 \neq 0},\ \ldots$$

$$\Sigma_2 \equiv \quad \ldots,\underbrace{\underline{x_2 = 1}}_{\alpha},\ \underline{x_3 \neq 2},\ x_6 \neq 1,\ x_8 \neq 3,\ \ldots$$

Algorithm 2: checkNegativeDecisions(Σs : set of increasing-nogoods)

```
    Data: Increasing-nogoods in $\Sigma$s have a common $\alpha$
1   foreach $x \in \bigcup_{\Sigma \in \Sigma s}$ diffVars($\Sigma$) do
2       if $dom(x) \subseteq \bigcup_{\Sigma \in \Sigma s}$ diffValues($\Sigma, x$) then
3           falsify $\alpha(\Sigma)$ ;                    // $\Sigma$ can be any increasing-nogood from $\Sigma$s
```

and let us assume that all variables have the same domain $\{0,1,2,3\}$. On this example, we can observe that $x_2 = 1$ is the common α to this group of three increasing-nogoods. By looking at the negative decisions following these three occurrences of α (the precise values of β are not relevant for our illustration), we can collect $\{0,1,2\}$ as values involved in watched negative decisions for x_3 (they are necessarily put before β which is not represented here). This means that if x_2 is assigned the value 1 then the only remaining value in $dom(x_3)$ will be 3. On the other hand, if at a certain moment, the domain of x_3 does not contain anymore the value 3, it is absolutely necessary to prevent x_2 from being assigned the value 1. □

Algorithm 2 is a generalization of Algorithm 1, by considering groups of increasing-nogoods instead of increasing-nogoods individually. Algorithm 2 has a worst-case time complexity in $O(nd+g)$ where g is the sum of the size of the increasing-nogoods in Σs (we have $g = \sum_{\Sigma \in \Sigma s} |\Sigma|$). Indeed, precomputing sets $\bigcup_{\Sigma \in \Sigma s}$ diffVars(Σ) and $\bigcup_{\Sigma \in \Sigma s}$ diffValues(Σ, x) can be performed in $O(g)$ by scanning every decision in the increasing-nogoods of Σs. With these precomputed sets, executing lines 1–2 is in $O(nd)$.

4.3 Combining Increasing Nogoods using Pivots

We call *pivot* a variable x such that for any value $a \in dom(x)$ there exists an increasing-nogood Σ such that $\alpha(\Sigma)$ is the positive decision $x = a$; in that case, we say that Σ is a *support* of pivot x for a. Interestingly, once a pivot variable x is identified, it is possible to infer negative decisions that are shared by all supports of x. This is the principle of the algorithm we present after an illustration.

Example 4.3. *Let us consider the following three increasing-nogoods:*

$$\Sigma_0 \equiv \quad \ldots,\ x_6 \neq 2,\ \underbrace{x_2 = 1}_{\alpha},\ x_1 \neq 0,\ \underline{x_3 \neq 1},\ \ldots$$

$$\Sigma_1 \equiv \quad \ldots, x_7 \neq 0,\ x_1 \neq 2,\ \underbrace{x_2 = 0}_{\alpha},\ \underline{x_3 \neq 1},\ \ldots$$

$$\Sigma_2 \equiv \quad \ldots, \underbrace{x_2 = 2}_{\alpha},\ \underline{x_3 \neq 1},\ x_6 \neq 1,\ x_8 \neq 2,\ \ldots$$

and let us assume that all variables have the same domain $\{0,1,2\}$. *On this example, we can see that* x_2 *is a pivot since all its possible values are involved in* α *of different increasing-nogoods. As* $x_3 \neq 1$ *is a negative decision that is watched in the three increasing-nogoods, we can deduce that* x_3 *must always be different from* 1. □

Algorithm 3 implements the use of pivot variables for making additional inferences. Line 1 only iterates over the variables that are involved in some α (i.e., α of some increasing-nogoods). Line 2 tests if the variable x is indeed a pivot. Line 3 iterates over the decisions that are shared by all supports of x. Each such decision must be forced to be satisfied. Note that an optimization consists in only checking that a decision is shared by some subsets of supports of x, the subsets with exactly one support of x for each value. Algorithm 3 has a worst-case time complexity in $O(n^2dp)$ where p is the number of increasing-nogoods.

Algorithm 3: checkPivots(Σs : the full set of increasing-nogoods)

```
1 foreach $x \in \{var(\alpha(\Sigma)) : \Sigma \in \Sigma s\}$ do
2     if $dom(x) \subseteq \{val(\alpha(\Sigma)) : \Sigma \in \Sigma s \wedge var(\alpha(\Sigma)) = x\}$ then
3         foreach $\delta \in \bigcap\{\Sigma \in \Sigma s : var(\alpha(\Sigma)) = x\}$ do
4             satisfy $\delta$;
```

5 Experiments

We have conducted an experimentation on a computer *Intel Xeon X5550* clocked at 2,67 GHz and equipped with 8 GBytes of RAM. Our initial benchmark was composed of all instances that were used during XCSP 2.1 solver competitions. We discarded the series of instances that were either too easy to solve (less than 1 second) or too hard to solve (more than 900 seconds) when employing MAC without nogood recording; this yielded a set composed of $3,744$ instances. For our experiments, we have used the solver *rclCSP* introduced in [6]. The variable ordering heuristic is *dom/wdeg* [1] and the restart policy corresponds to a geometric series of first term 10 and common ratio 1.1. Given the complexity of Algorithm 3 and because Algorithm 1 is generalized by Algorithm 2, we chose to conduct our experiments with only Algorithm 2 (combining increasing-nogoods of similar α). For our comparison, we tested three methods: NRR (nogood recording from restarts as proposed in [12]), IncNG (managing

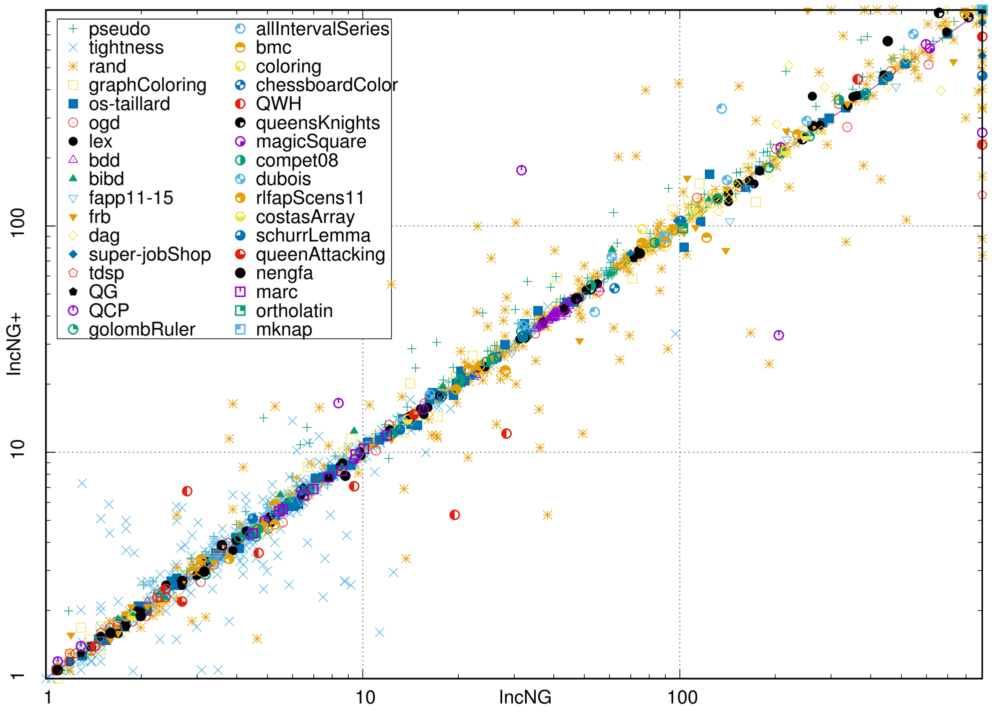

Figure 2: Pairwise comparison (CPU time in seconds) of IncNG and IncNG+. Results obtained on the $3,744$ instances used as benchmarks. The timeout to solve an instance is set to 900 seconds.

Series	#inst	NLD		IncNG		IncNG+	
		#sols	PAR10	#sols	PAR10	#sols	PAR10
costasArray	11	9	1,745	**9**	**1,686**	**9**	**1,686**
fapp11-15	55	42	2,162	42	2,157	**42**	**2,155**
frb45-21	10	9	1,138	**10**	**232**	10	282
nengfa	10	9	970	**9**	**954**	9	974
ogdVg	65	35	4,197	39	3,637	**40**	**3,511**
ortholatin	9	**4**	**5,045**	3	6,011	3	6,010
os-taillard-7	30	**19**	**3,353**	18	3,679	18	3,680
QCP-20	15	6	5,437	7	4,870	**8**	**4,289**
QWH-20	10	10	79	**10**	**47**	10	48
QWH-25	10	1	8,187	0	9,000	**2**	**7,291**
rand-2-50-23-fcd	50	5	8,146	12	6,953	**15**	**6,436**
rand-2-50-23	50	4	8,335	9	7,476	**9**	**7,462**
rand-3-24-24	50	14	6,596	16	6,248	**18**	**5,900**
rlfapScens11	12	11	852	**11**	**844**	11	852
super-jobShop	46	34	2,358	33	2,547	**35**	**2,186**

Table 1: Average results on 15 series of instances (timeout set to 900 seconds).

increasing-nogoods as proposed in [10]) and IncNG+ (combining increasing-nogoods, our approach as introduced in section 4.2). Figure 2 displays the overall results that we have obtained. The scatter plot shows that our approach (IncNG+) has usually a small overhead (when it turns out to be not very effective), and interestingly makes search a little bit more robust (see the dots on the right vertical line that corresponds to unsolved instances by IncNG). Table 1 shows a detailed comparison between the three methods on some series. The table contains the following information: the name of the series, the number of instances in the series (*#inst*) and for the three tested approaches, the number of solved instances (*#sols*) and the PAR10 score that is the average of the runtimes while considering 10 times the timeout (900) for unsolved instances. In general, our approach (IncNG+) solves at least as many instances as IncNG and sometimes more (see, for example, *super-jobShop*). When IncNG and IncNG+ both solve the same number of instances, we usually notice a slight loss for our approach due to the management of the partitions of the increasing-nogoods (for example, see *frb45-21*). But interestingly, there are series where this processing time is compensated by a better pruning of the search tree, with a substantial time saving as outcome (*rand-2-50-23*). Moreover, we observe that, in general, the more difficult the instance is, the more competitive our approach is, as illustrated by the *QWH-20* and *QWH-25* series which admit increasing sizes and complexities.

6 Conclusion

In this paper, we have introduced three general rules allowing us to reason with (groups of) increasing-nogoods. We have shown experimentally the practical interest of the second rule (which generalizes the first one): when it is effective, i.e., when inferences can be performed by reasoning on groups of increasing-nogoods, it saves computation time, and when it is not effective, the overhead is rather limited. We believe that some improvements are still possible, especially on the exploitation of pivot variables.

Acknowledgement

This work has been supported by the project CPER DATA from the ”Hauts-de-France” Region.

References

[1] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 146–150. IOS Press, 2004.

[2] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[3] Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 352–366. Springer, 2009.

[4] Daniel Frost and Rina Dechter. Dead-end driven learning. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 294–300. AAAI Press / The MIT Press, 1994.

[5] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research (JAIR)*, 1:25–46, 1993.

[6] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. A CSP solver focusing on fac variables. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 493–507. Springer, 2011.

[7] Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*, pages 249–261. Springer, 2000.

[8] George Katsirelos and Fahiem Bacchus. Unrestricted nogood recording in CSP search. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 873–877. Springer, 2003.

[9] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Nogood recording from restarts. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 131–136, 2007.

[10] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1(3-4):147–167, 2007.

[11] Christophe Lecoutre and Sebastien Tabary. Symmetry-reinforced Nogood Recording from Restarts. In *11th International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon'11)*, pages 13–27, Perugia, Italy, 2011.

[12] Jimmy H. M. Lee, Christian Schulte, and Zichen Zhu. Increasing nogoods in restart-based search. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 3426–3433. AAAI Press, 2016.

[13] Jimmy H. M. Lee and Zichen Zhu. An increasing-nogoods global constraint for symmetry breaking during search. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 465–480. Springer, 2014.

[14] Thomas Schiex and Gérard Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools*, 3(2):187–208, 1994.