Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Increasing Nogoods in Restart-Based Search*

Jimmy H. M. Lee,† Christian Schulte,‡ and Zichen Zhu†
†Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{jlee,zzhu}@cse.cuhk.edu.hk
‡School of ICT, KTH Royal Institute of Technology, Sweden
cschulte@kth.se

Abstract

Restarts are an important technique to make search more robust. This paper is concerned with how to maintain and propagate nogoods recorded from restarts efficiently. It builds on reduced nld-nogoods introduced for restarts and increasing nogoods introduced for symmetry breaking. The paper shows that reduced nld-nogoods extracted from a single restart are in fact increasing, which can thus benefit from the efficient propagation algorithm of the incNGs global constraint. We present a lighter weight filtering algorithm for incNGs in the context of restart-based search using dynamic event sets (dynamic subscriptions). We show formally that the lightweight version enforces GAC on each nogood while reducing the number of subscribed decisions. The paper also introduces an efficient approximation to nogood minimization such that all shortened reduced nld-nogoods from the same restart are also increasing and can be propagated with the new filtering algorithm. Experimental results confirm that our lightweight filtering algorithm and approximated nogood minimization successfully trade a slight loss in pruning for considerably better efficiency, and hence compare favorably against existing state-of-the-art techniques.

Introduction

Restarts are an important technique to make search more robust (Dechter 1990; Frost and Dechter 1994; Schiex and Verfaillie 1994). The benefit of restarts can be considerably enhanced by recording nogoods generated at the end of each search run (restart point) and propagating them in future search runs to avoid repeating erroneous search decisions. Lecoutre *et al.* (2007) extract a set of reduced nld-nogoods at each restart point and show how such a set of reduced nld-nogoods can be propagated using watched literals (Moskewicz et al. 2001). However, such reduced nld-nogoods are abundant and weak in constraint propagation.

In this paper we exploit the observation that reduced nld-nogoods collected from a restart are in fact increasing, allowing all such nogoods to be processed by one incNGs global constraint (Lee and Zhu 2014). We also prove that the structure of a set of increasing nogoods can be obtained from a restart-based search engine for free.

*This research has been supported by the grant CUHK413713 from the Research Grants Council of Hong Kong SAR.

The incNGs global constraint filtering algorithm (Lee and Zhu 2014) supports *dynamic* addition of nogoods to the increasing nogoods set which is required for dynamic symmetry breaking. There is no such need for nogoods computed from restarts. To trade pruning power for efficiency, a lightweight version of the incNGs global constraint filtering algorithm is proposed. Without the requirement to be dynamic for restarts, this filtering algorithm further utilizes dynamic subscriptions (dynamic event sets (Schulte and Stuckey 2008), also known as dynamic triggers (Gent, Jefferson, and Miguel 2006)), an adaptation of watched literals (Moskewicz et al. 2001), to improve efficiency. Dynamic subscriptions decrease unnecessary activations of the filtering algorithm during constraint propagation. While the lightweight version requires fewer subscriptions, it has the same consistency level as GAC on each nogood.

Lecoutre *et al.* (2007) also propose minimal reduced nld-nogoods with respect to an inference operator. Minimal reduced nld-nogoods are shorter and hence offer more propagation. Unfortunately, minimal reduced nld-nogoods generated from a restart are not increasing. We therefore introduce an efficient approximation to reduced-nld nogood minimization such that all shortened reduced nld-nogoods from the same restart are a set of increasing nogoods. Experimental evaluation demonstrates the advantage of our global constraint filtering algorithm and shortening method.

Background

A *constraint satisfaction problem* (CSP) P is a tuple (X, D, C) where X is a finite set of variables, D is a finite set of domains such that each $x \in X$ has a domain $D(x)$ and C is a set of constraints, each is a subset of the Cartesian product $D(x_{i_1}) \times \cdots \times D(x_{i_k})$ of the domains of the involved variables (*scope*). A constraint is *generalized arc consistent (GAC)* iff when a variable in the scope of a constraint is assigned any value in its domain, there exist compatible values (called *supports*) in the domains of all the other variables in the scope of the constraint. A CSP is *GAC* iff every constraint is GAC. An *assignment* assigns a value v to a variable x. A *full assignment* is a set of assignments, one for each variable in X. A *solution* to P is a full assignment that satisfies every constraint in C.

A *decision* can be either +ve or -ve. A *+ve decision* is an equality constraint $x = v$ and a *-ve decision* has the form

$x \neq v$. The variable x *interferes* with decisions $x = v$ and $x \neq v$. A +ve decision $x = v$ is *satisfied* (resp. *falsified*) iff $D(x) = \{v\}$ (resp. $v \notin D(x)$). A decision is *satisfied* (resp. *falsified*) if its negation is falsified (resp. satisfied); otherwise the decision is *unsatisfied* (resp. *unfalsified*).

A *nogood* is the negation of a conjunction (or set) of +ve decisions which is not compatible with any solution. A nogood is *satisfied* iff one of its +ve decisions is falsified. Nogoods can also be expressed in an equivalent implication form. A *directed nogood* ng is an implication of the form $(x_{s_1} = v_{s_1}) \wedge \cdots \wedge (x_{s_m} = v_{s_m}) \Rightarrow (x_k \neq v_k)$ with the *left hand side* (LHS) $\mathrm{lhs}(ng) \equiv (x_{s_1} = v_{s_1}) \wedge \cdots \wedge (x_{s_m} = v_{s_m})$ and the *right hand side* (RHS) $\mathrm{rhs}(ng) \equiv (x_k \neq v_k)$. The meaning of ng is that the negation of the RHS is incompatible with the LHS, and v_k should be pruned from $D(x_k)$ when the LHS is satisfied.

We use dynamic subscriptions (dynamic event sets (Schulte and Stuckey 2008), also known as dynamic triggers (Gent, Jefferson, and Miguel 2006)) for implementing filtering algorithms for a nogood or a conjunction of nogoods. In the context of directed nogoods, when a +ve (resp. -ve) decision is *subscribed* by a filtering algorithm, the filtering algorithm would be *triggered* if the decision is satisfied (resp. falsified). Decisions in directed nogoods are dynamically subscribed and unsubscribed during filtering. This is an adaption of the watched literal technique (Moskewicz et al. 2001) which is key for efficient implementation of SAT solvers. The difference is decisions subscribed by dynamic subscriptions are backtrackable while decisions are backtrack-stable for watched literals (Gent, Jefferson, and Miguel 2006).

We consider binary search trees, in which every non-leaf node has exactly two children. Suppose a non-leaf node P_1 has x and $v \in D(x)$ as the branching variable and value. The left and right children of P_1 are $P_1 \cup \{x = v\}$ and $P_1 \cup \{x \neq v\}$ respectively. We call $x = v$ the *decision from* P_1 *to* $P_1 \cup \{x = v\}$ and $x \neq v$ is the *decision from* P_1 *to* $P_1 \cup \{x \neq v\}$. A *branch* from root P to a node P_1 is the sequence of all decisions from P to P_1.

A set of (directed) nogoods Λ is *increasing* (Lee and Zhu 2014) if the nogoods can form a sequence $\langle ng_0, \ldots, ng_t \rangle$ where $ng_i \equiv (A_i \Rightarrow x_{k_i} \neq v_{k_i})$ such that (i) for any $i \in [1, t]$, $A_{i-1} \subseteq A_i$ and (ii) no nogoods are implied by another. A nogood ng_i in Λ is *lower than* nogood ng_j iff $i < j$, and ng_j is *higher than* ng_i. The increasing nogoods global constraint, incNGs, and its filtering algorithm are proposed for a set of increasing nogoods (Lee and Zhu 2014).

Nogoods in Restart-Based Search

Lecoutre *et al.* (2007) give a nogood recording framework in restart-based search to avoid searching the same regions from one restart to another. They extract nogoods from the current branch of the search tree at the end of each restart and exploit these nogoods in subsequent search runs. In the following, we show that the extracted nogoods are increasing and thus the incNGs global constraint can be adopted.

Given a branch, which is a sequence of decisions $\Sigma = \langle \delta_0, \ldots, \delta_{m-1} \rangle$. The subsequence $\langle \delta_0, \ldots, \delta_i \rangle$, where δ_i is a -ve decision, is called an *nld-subsequence* (-ve last decision subsequence) of Σ. The set of +ve and -ve decisions of Σ are denoted by $pos(\Sigma)$ and $neg(\Sigma)$ respectively. Lecoutre *et al.* (2007) prove that, given a branch Σ of the search tree, for any nld-subsequence $\Sigma' = \langle \delta_0, \ldots, \delta_i \rangle$, the negation of the set $pos(\Sigma') \cup \{\neg\delta_i\}$ is a nogood (*reduced nld-nogood*).

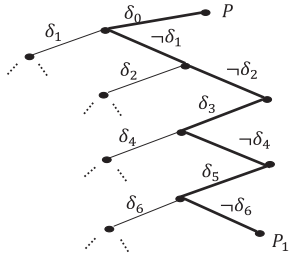

Figure 1: Partial search tree

Figure 1 depicts a partial search tree. Each left or right branch is labeled by its decision (δ_i or $\neg\delta_i$). The highlighted branch from root P to node P_1 is the sequence $\Sigma = \langle \delta_0, \neg\delta_1, \neg\delta_2, \delta_3, \neg\delta_4, \delta_5, \neg\delta_6 \rangle$. All nld-subsequences of Σ are $\langle \delta_0, \neg\delta_1 \rangle$, $\langle \delta_0, \neg\delta_1, \neg\delta_2 \rangle$, $\langle \delta_0, \neg\delta_1, \neg\delta_2, \delta_3, \neg\delta_4 \rangle$ and $\langle \delta_0, \neg\delta_1, \neg\delta_2, \delta_3, \neg\delta_4, \delta_5, \neg\delta_6 \rangle$. Their corresponding reduced nld-nogoods are $\neg\{\delta_0, \delta_1\}$, $\neg\{\delta_0, \delta_2\}$, $\neg\{\delta_0, \delta_3, \delta_4\}$ and $\neg\{\delta_0, \delta_3, \delta_5, \delta_6\}$.

For a reduced nld-nogood $\Delta = \neg\{\delta_0, \ldots, \delta_i\}$, the *directed reduced nld-nogood* of Δ is $\delta_0 \wedge \cdots \wedge \delta_{i-1} \Rightarrow \neg\delta_i$. The set of directed reduced nld-nogoods extracted from the highlighted branch of Figure 1 are:

$$\begin{array}{ll} \delta_0 & \Rightarrow \neg\delta_1 \\ \delta_0 & \Rightarrow \neg\delta_2 \\ \delta_0 \wedge \delta_3 & \Rightarrow \neg\delta_4 \\ \delta_0 \wedge \delta_3 \wedge \delta_5 & \Rightarrow \neg\delta_6 \end{array} \tag{1}$$

Obviously, they are increasing, and hence we have:

Lemma 1. *The set of all directed reduced nld-nogoods ng extracted from a branch $\Sigma = \langle \delta_0, \ldots, \delta_{m-1} \rangle$ are increasing.*

We propose a more natural compact encoding than that by Lee and Zhu (2014) for restart-based search. A sequence of increasing nogoods

$$\begin{array}{rl} ng_0 = & \delta_{s_{00}} \wedge \cdots \wedge \delta_{s_{0r}} \Rightarrow \neg\delta_{k_0} \\ ng_1 = & \mathrm{lhs}(ng_0) \wedge \delta_{s_{10}} \wedge \cdots \wedge \delta_{s_{1r}} \Rightarrow \neg\delta_{k_1} \\ \vdots & \vdots \\ ng_t = & \mathrm{lhs}(ng_{t-1}) \wedge \delta_{s_{t0}} \wedge \cdots \wedge \delta_{s_{tr}} \Rightarrow \neg\delta_{k_t} \end{array} \tag{2}$$

can be encoded compactly as a sequence of decisions

$$\begin{array}{rll} \Sigma = \langle & \delta_{s_{00}}, & \ldots, \delta_{s_{0r}}, \neg\delta_{k_0}, \\ & \delta_{s_{10}}, & \ldots, \delta_{s_{1r}}, \neg\delta_{k_1}, \\ & \vdots & \\ & \delta_{s_{t0}}, & \ldots, \delta_{s_{tr}}, \neg\delta_{k_t} \ \rangle \end{array} \tag{3}$$

We call Σ the *encoding sequence* for $\langle ng_0, \ldots, ng_t \rangle$. The advantage of this encoding is three-fold. First, it is compact, avoiding storing same decisions repeatedly. Second, it also

avoids redundant checks of the same decisions, especially in verifying pruning conditions as explained in the following section. Third, the encoding can be readily extracted from the search branch up to the current node, which follows directly from the definition and is shown in the next theorem.

Theorem 1. *Given a branch $\Sigma = \langle \delta_0, \ldots, \delta_{m-1} \rangle$ and the extracted increasing nogoods $\Lambda = \langle ng_0, \ldots, ng_t \rangle$. Σ is the encoding sequence for Λ.*

The increasing nogoods in (1) are encoded as $\langle \delta_0, \neg\delta_1, \neg\delta_2, \delta_3, \neg\delta_4, \delta_5, \neg\delta_6 \rangle$, which is exactly the branch from root P to node P_1 in Figure 1. Thus, the encoding can be obtained directly from the restart-based search engine, without any dedicated construction.

A Lightweight Filtering Algorithm

Lee and Zhu (2014) give a filtering algorithm for the incNGs global constraint. This algorithm is *dynamic* in the sense that the increasing nogoods set is empty at the root node and symmetry breaking nogoods are added dynamically during search at every backtrack point. Once a new nogood is generated and added to its corresponding increasing nogoods set, the filtering algorithm needs to be triggered.

In restarts, a set of increasing nogoods is extracted at each restart. An incNGs global constraint is therefore posted to filter this set of increasing nogoods in subsequent search runs. No new nogoods are added to this set of increasing nogoods. Without the requirement to be dynamic for restarts, dynamic subscriptions can thus be used to improve the efficiency. Moreover, even though the filtering algorithm by Lee and Zhu (2014) provides stronger propagation than GAC on each nogood, the overhead is considerable while additional prunings are rare. Therefore, we propose a lightweight filtering algorithm, LightFilter, with dynamic subscriptions.

To enforce GAC on a nogood which is the negation of a conjunction of +ve decisions, two unsatisfied decisions need to be subscribed. As long as both subscribed decisions are not satisfied, this nogood is GAC. Once one of the decisions is satisfied and there are no other unsatisfied decisions, the other subscribed decision must be enforced to be false and the nogood is satisfied. For each nogood in the increasing nogoods set, LightFilter also only subscribes to two unsatisfied decisions and only prunes values according to the above rule. Utilizing the structure of the encoding sequence, we now optimize the subscribed decisions set for increasing nogoods.

Since the LHSs of higher nogoods subsume those of lower nogoods for increasing nogoods and a nogood must be GAC if it has two unsatisfied +ve decisions, we can formulate the following lemma.

Lemma 2. *Given a set of increasing nogoods $\Lambda = \langle ng_0, \ldots, ng_t \rangle$. Suppose ng_i has two unsatisfied decisions in its LHS. Then the nogoods ng_j where $j \geq i$ are GAC.*

Thus we only need to subscribe to the first two unsatisfied +ve decisions δ_α and δ_β in the encoding sequence to enforce GAC on all nogoods containing them.

Consider the following set of increasing nogoods

$$\begin{aligned} ng_0 \equiv \quad & x_2 = 1 \Rightarrow x_3 \neq 1, \\ ng_1 \equiv \quad & x_2 = 1 \wedge x_4 = 1 \Rightarrow x_1 \neq 1, \\ ng_2 \equiv \quad & x_2 = 1 \wedge x_4 = 1 \wedge x_5 = 1 \Rightarrow x_6 \neq 2 \end{aligned} \tag{4}$$

with $D(x_1) = \cdots = D(x_6) = \{1, 2\}$. The encoding sequence is $\Sigma = \langle x_2 = 1, x_3 \neq 1, x_4 = 1, x_1 \neq 1, x_5 = 1, x_6 \neq 2 \rangle$. Subscribing to $\delta_\alpha \equiv (x_2 = 1)$ and $\delta_\beta \equiv (x_4 = 1)$ is enough to enforce GAC on ng_1 and ng_2.

Suppose nogood ng only contains δ_α but not δ_β (e.g. ng_0). If its RHS is falsified, δ_α must be enforced to be false and now all nogoods containing δ_α are satisfied. Suppose $D(x_3) = \{1\}$, $lhs(ng_0)$ is enforced to be false and all three nogoods in (4) are satisfied. Otherwise, we also subscribe to this unfalsified RHS. Subscribing to $\delta_\alpha \equiv (x_2 = 1)$ and $rhs(ng_0) \equiv (x_1 \neq 1)$ is enough to ensure ng_0 is GAC with $D(x_1) = \cdots = D(x_6) = \{1, 2\}$ in (4). Suppose a nogood does not contain δ_α. Its LHS must be true and its RHS can be directly enforced to be true. In conclusion, when an incNGs is posted, its filtering algorithm LightFilter *subscribes initially to the first two unsatisfied +ve decisions and all unfalsified -ve decisions between them.*

Updating the subscribed decisions during filtering is similar to that of enforcing GAC on a nogood (Moskewicz et al. 2001). Here we scan the encoding sequence from left to right to find subscribed decisions. The LightFilter filtering algorithm is triggered in three cases:

(a) The first subscribed +ve decision δ_α is satisfied. The RHSs of all nogoods ng, whose LHSs only contain δ_α but not δ_β, are enforced to be true. Now only δ_β is subscribed for the remaining nogoods. Thus we need to find the next unsatisfied +ve decision to subscribe by scanning the encoding sequence from δ_β to the right. If we encounter a -ve decision δ_i during scanning, the nogood with δ_i as RHS only has one unsatisfied decision δ_β in its LHS. If δ_i is falsified, the current δ_β is enforced to be false and all nogoods are satisfied. Otherwise, this unfalsified δ_i is subscribed.

(b) A subscribed -ve decision δ_γ is falsified. Now δ_α is enforced to be false and all nogoods are satisfied.

(c) The second subscribed +ve decision δ_β is satisfied. Now all nogoods containing δ_β only have one decision δ_α being subscribed. Thus we need to find the next unsatisfied +ve decision to subscribe as in case (a).

We propose LightFilter on an encoding sequence Σ for a set of increasing nogoods and a CSP $P = (X, D, C)$. Neg(δ) returns true if δ is a -ve decision.

When an incNGs constraint is posted, all nogoods with LHSs empty or true are directly enforced. After that, the first two +ve decisions δ_α and δ_β and all -ve decisions between them are subscribed and Algorithm 1 is immediately triggered once. Algorithm 1 updates α and β using *UpdateAlpha* (line 1) and *UpdateBeta* (line 2) respectively. If all nogoods are satisfied ($m = 0$), the incNGs constraint is deleted (line 3).

To update α in Algorithm 2, we check the current subscribed decision at α repeatedly until it is not satisfied (line

Algorithm 1 *LightFilter*()

Require:
Σ: the encoding sequence
$m = |\Sigma|$
α, β: the position of the two leftmost unsatisfied +ve decisions

1: *UpdateAlpha*();
2: **if** $m \neq 0 \wedge \beta \neq m$ **then** *UpdateBeta*();
3: **if** $m = 0$ **then** delete constraint;

10). If it is satisfied (case (a)), we need to satisfy the RHSs of nogoods whose LHSs have δ_α but not δ_β (lines 3-5). If there are no decisions left (line 6), all nogoods are satisfied. Otherwise, we set α to β (line 7) and call *WatchFollowDec* (line 8) which finds the next +ve decision δ_i and subscribes to all unfalsified -ve decisions between δ_α and δ_i. If a falsified -ve decision is encountered along the subscription in *WatchFollowDec*, δ_α is enforced to be false and all nogoods are satisfied. Otherwise, we check the updated subscribed decision at α again (line 9). After ensuring δ_α is not satisfied (line 10), we need to check all subscribed -ve decisions for case (b). If one of them is falsified (line 12), δ_α is enforced to be false (line 13) and all nogoods are satisfied (line 14).

Algorithm 2 *UpdateAlpha*()

1: **if** δ_α is satisfied **then**
2: unsubscribe δ_α;
3: **for** $i \in [\alpha + 1, \beta)$ **do**
4: **if** Neg(δ_i) **then**
5: satisfy δ_i, unsubscribe δ_i;
6: **if** $\beta = m$ **then** $m = 0$; **return**;
7: $\alpha = \beta$;
8: *WatchFollowDec*(α);
9: **if** $m \neq 0$ **then** *UpdateAlpha*();
10: **else**
11: **for** $i \in [\alpha + 1, \beta)$ **do**
12: **if** Neg(δ_i) $\wedge$ δ_i is falsified **then**
13: falsify δ_α;
14: $m = 0$; **return**;

After ensuring δ_α is not satisfied, we update β in Algorithm 3. Similar to updating α, we check the current subscribed decision at β repeatedly until it is not satisfied. If it is satisfied (case (c)) (line 1), we update it using *WatchFollowDec* (line 3).

Algorithm 3 *UpdateBeta*()

1: **if** δ_β is satisfied **then**
2: unsubscribe δ_β;
3: *WatchFollowDec*(β);
4: **if** $m \neq 0$ **then** *UpdateBeta*();

We have the following lemma for the subscribed decisions in a set of increasing nogoods after updating of LightFilter.

Lemma 3. *Given an encoding sequence Σ for a set of increasing nogoods. LightFilter always subscribes to the first two unsatisfied +ve decisions and all unfalsified -ve decisions between them.*

LightFilter makes all nogoods ng which are not true yet have either (i) two unsatisfied +ve decisions δ_α and δ_β or (ii) one unsatisfied +ve decision δ_α and one unfalsified -ve decision $\mathrm{rhs}(ng)$ being subscribed. The following theorem holds.

Theorem 2. *The consistency level of LightFilter on a set of increasing nogoods is equivalent to GAC on each nogood.*

Since each decision in the encoding sequence Σ is scanned at most once when LightFilter is triggered, the time complexity of LightFilter is as follows.

Theorem 3. *Given a CSP $P = (X, D, C)$. The time complexity of LightFilter is $O(\sum_{x \in X} |D(x)|)$.*

Proof. The worst case is that all decisions in Σ are checked and the maximum size of Σ is $\sum_{x \in X} |D(x)|$. □

Towards Minimal Nogoods

Shorter nogoods provide stronger pruning. Lecoutre *et al.* (2007) propose minimal reduced nld-nogoods with respect to an inference operator ϕ which is used for enforcing a consistency level at each node. The negation of a set of decisions Δ is a *ϕ-nogood* of a CSP $P = (X, D, C)$ iff after applying ϕ to $P \cup \Delta$, there exists a variable x in X where $|D(x)| = 0$. The negation of Δ is a *minimal ϕ-nogood* of a CSP P iff there does not exist $\Delta' \subset \Delta$ such that $\neg\Delta'$ is a ϕ-nogood. Given a reduced nld-nogood ng generated from an nld-nogood which is also a ϕ-nogood. Lecoutre *et al.* generate a minimal nogood using a constructive approach (Moskewicz et al. 2001) by finding all decisions which clearly belong to a minimal ϕ-nogood (*transition decisions*).

Minimizing a set of increasing nogoods might not result in increasing nogoods. For example, given a set of increasing nogoods $\{ng_0, \ldots, ng_t\}$, the LHS of a minimal nogood of ng_0 includes the first two +ve decisions while the LHS of a minimal nogood of ng_1 includes the last two +ve decisions which do not necessarily subsume the first two.

We now give an approximation of the minimization for a set of increasing ϕ-nogoods where each computed nogood is a ϕ-nogood but not necessarily minimal. We shorten the nogoods from lowest to highest in the increasing nogoods sequence. After the first k nogoods are shortened, to shorten the $(k+1)$-st one, we make sure the structure of the previous k shortened nogoods is kept. Thus we find transition decisions only from $(\mathrm{lhs}(ng_{k+1}) - \mathrm{lhs}(ng'_k)) \cup \neg\,\mathrm{rhs}(ng_{k+1})$ where ng'_k is the shortened nogood of ng_k.

Given a CSP $P = (X, D, C)$, an encoding sequence Σ for a set of increasing ϕ-nogoods and an inference operator ϕ, we use the following approach to do the shortening.

For each nogood ng from lowest to highest (line 2), Algorithm 4 does not minimize the entire nogood ng. Instead, it only chooses transition decisions from the decisions in $lhs(ng)$ that are not contained in the LHS of the last shortened nogood, as well as the negation of $rhs(ng)$ (line 4).

Algorithm 4 *ApproxConstructive*(P,Σ,ϕ)

Require:
$\Delta = \emptyset$: the set of transition decisions
$\alpha = -1$: the position of the current last -ve decision
1: $P' = P$;
2: **for** $i \in [0, |\Sigma| - 1]$ **do**
3: **if** δ_i is a -ve decision **then**
4: $\Delta =$ *Constructive*(P',ϕ,$\{\delta_{\alpha+1},\ldots,\neg\delta_i\}$);
5: reorder Σ from $\alpha + 1$ to i to:
6: put $\Delta - \{\neg\delta_i\}$ into the first place,
7: δ_i into the second place, and
8: the remaining into the third place;
9: $\alpha = \alpha + |\Delta|$;
10: $P' = P' \cup (\Delta - \{\neg\delta_i\})$;

Constructive(P',ϕ,ng') (line 4) returns a set of transition decisions Δ that comprises a minimal nogood of ng'. After finding Δ, we reorder Σ from the position of the last -ve decision accordingly (lines 5-8). Now α is set to the position of the current -ve decision after reordering (line 9).

Since nogoods in the encoding sequence resulting from Algorithm 4 are *shortened nogoods* of the original set of increasing nogoods, we have the following theorem.

Theorem 4. *Given a CSP $P = (X, D, C)$, an inference operator ϕ and an encoding sequence Σ. The shortened nogoods generated by Algorithm 4 are increasing.*

We now give the soundness and the time complexity of our algorithm without proof due to space limitation.

Theorem 5. *Given a CSP P, an inference operator ϕ and an encoding sequence Σ. The nogoods in the encoding sequence computed by Algorithm 4 are ϕ-nogoods.*

An inference operator ϕ is *incremental* (Lecoutre et al. 2007) if the worst case time complexity of applying ϕ to a CSP from two respective sets of decisions Δ' and Δ such that $\Delta' \subset \Delta$ are equivalent. For example, all (known) GAC inference algorithms are incremental.

Theorem 6. *Given a CSP $P = (X, D, C)$, an incremental inference operator ϕ and an encoding sequence Σ. The time complexity of Algorithm 4 is $O(\sum_{x \in X} |D(x)|\xi)$ where ξ is the cost of enforcing ϕ on P once.*

However, this approach is only an approximation in order to keep the increasing structure.

Observation 1. *Given a CSP P, the inference operator ϕ and an encoding sequence Σ. Nogoods in the encoding sequence computed by Algorithm 4 are not necessarily minimal.*

Experimental Evaluation

This section gives three sets of experiments to demonstrate the advantage of using incNGs in restarts and the dynamic subscription based lightweight filtering algorithm. Two of these experiments have been used to show the advantage of nogoods recording (Lecoutre et al. 2007) as well. All experiments are conducted using Gecode 4.3.2 on Intel machines with 30-45GB of memory and 64-bit Debian 6.0.6 operating system. Due to the number of experiments performed, different machines were used for different sets of problems. When a decision is subscribed by a filtering algorithm in Gecode, the filtering algorithm is triggered for execution when the domain of the interfering variable changes. In this way, satisified nogoods are immediately deleted. As a baseline we use search without restarts (NORST) and with restarts but without nogoods (RST) respectively. When exploiting nogoods recorded upon restarts, we propagate nogoods individually with dynamic subscriptions similar to that given by Lecoutre *et al.* (2007) (RST+NG) and the two filtering algorithms for the incNGs global constraint: FullFilter as given by Lee and Zhu (2014) (RST+NG$_G$) and our LightFilter (RST+NG$_{LG}$). We also present results exploiting minimal nogoods and the filtering algorithm given by Lecoutre *et al.* (2007) (RST+NG^M). We use RST+NG^m_G and RST+NG$^m_{LG}$ to denote results of exploiting shortened nogoods generated by Algorithm 4 using FullFilter (Lee and Zhu 2014) and LightFilter respectively.

We use the universal restarting strategy by Luby *et al.* (1993). The failure limit at each restart according to the Luby strategy is $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots)$ and we scale each element by a factor of 100. We have chosen Luby rather than a more aggressive restarting strategy such as geometric (aka Walsh) (Walsh 1999) to exercise the recording and propagation of nogoods.

We exploit the adaptive dom/wdeg (Boussemart et al. 2004) variable ordering heuristic as a simple and yet effective generic heuristic. In dom/wdeg, the weight of constraints are preserved after each restart. To make the comparison of different filtering algorithms as meaningful as possible, we ignore the weighted degree incurred by the respective nogoods filtering algorithm. Note that this still cannot ensure that the behavior of dom/wdeg is exactly the same under different filtering algorithms due to the essentially non-deterministic execution of filtering algorithms in Gecode (Schulte and Stuckey 2008). This issue never occurs for Queens Knights instances, but does so for several instances of the other two problems. We therefore also use the classical but less efficient dom/ddeg (Bessiere and Régin 1996) heuristic to solve Open Shop Scheduling. Similarly, to make the comparison among each filtering algorithm fair, we do not count the degree incurred by all nogoods filtering algorithms for the degree of variables. In this case, dom/ddeg is not affected by the nogoods filtering algorithms since the filtering algorithm ordering does not affect the degree of variables. In contrast, dom/wdeg attaches to each constraint a weight to compute the weighted degree of a variable. Such weights depend on how often the constraint is failed during search. In case more than one constraint can trigger failure during filtering, the execution order of filtering algorithms becomes important and can affect the weighted degrees of variables.

In our tables, $\#f$ denotes number of failures and t denotes runtime in seconds. The best results are highlighted in **bold**.

Queens Knights. The task is to put n queens and k knights on a chessboard of size $n \times n$ so that no two queens can

attack each other and all knights form a cyclic knights tour (Boussemart et al. 2004) where a square is not allowed to be shared by both a queen and a knight.

Table 1: Queens Knights without minimization/shortening

	NORST		RST		RST+NG		RST+NG$_G$		RST+NG$_{LG}$	
n	$\#f$	t	$\#f$	t	$\#f$	t	$\#f$	t	$\#f$	t
25	71,569	20.47	6,355	2.26	**5,221**	1.93	**5,221**	1.86	**5,221**	**1.68**
50	–	–	37,926	53.66	**23,723**	41.70	**23,723**	34.33	**23,723**	**26.66**
70	–	–	88,454	251.27	**55,270**	256.93	**55,270**	203.05	**55,270**	**124.72**
90	–	–	98,055	443.28	**79,588**	571.38	**79,588**	485.95	**79,588**	**329.53**

Table 2: Queens Knights with minimization/shortening

	RST+NG^M		RST+NG^m_G		RST+NG$^m_{LG}$	
n	$\#f$	t	$\#f$	t	$\#f$	t
25	1,540	**0.83**	**1,538**	0.84	**1,538**	**0.83**
50	9,833	**12.39**	**9,721**	12.63	**9,721**	12.52
70	**18,970**	**49.25**	24,757	57.42	24,757	57.25
90	**43,067**	184.68	43,246	184.33	43,246	**184.08**

We set $k = 5$ and n ranges from 25 to 90 which renders all problems unsatisfiable. The search time-out limit is 20 minutes. Results of timeout cases are indicated with "–". Table 1 shows the results. Exploiting nogoods in restarts reduces the search tree size to 72% on average. However, time is not saved for RST+NG due to the overhead of nogood propagation. After introducing the global constraint and the FullFilter algorithm, RST+NG$_G$ runs 1.17 times faster than RST+NG on average. Utilizing LightFilter with dynamic subscriptions, RST+NG$_{LG}$ runs 1.37 and 1.63 times faster than RST+NG$_G$ and RST+NG on average. The number of collected nogoods ranges from 507 of easier problems to 8442 of harder problems. The results are in line with the prediction that the benefit of our methods are more prominent with more nogoods to process.

Comparing Table 1 and Table 2, it is easy to conclude that the search efficiency is considerably improved by exploiting shorter nogoods. Table 2 also shows that our shortened nogoods do not lose too much pruning as compared to the minimal ones. It is important to note the runtimes are similar since each instance has only 4 to 30 nogoods after minimization/shortening since all knight variables are singleton arc inconsistent (Lecoutre et al. 2007). Moreover, it takes 2% of runtime to do minimization/shortening for the hardest instance.

Open Shop Scheduling. We take Open Shop Scheduling from the 2006 CSP Solver Competition[1]. There are six sets of 30 instances. We again set a timeout limit of 20 minutes to find a solution. We give the number of solved instances out of the total 30 for each set, and also show the average time and average number of recorded nogoods for the solved instances for the three filtering methods respectively. The method with the largest number of solved instances is considered to be the best and ties are broken by runtime.

[1]http://www.cril.univ-artois.fr/ lecoutre/benchmarks.html

Table 3: Open Shop Scheduling using dom/wdeg without minimization and with time limit

n		NORST	RST	RST+NG	RST+NG$_G$	RST+NG$_{LG}$
os-taillard-4	Average t	0.10	0.40	0.10	**0.06**	**0.06**
	# Solved	**30/30**	**30/30**	**30/30**	**30/30**	**30/30**
	# Nogoods			307	307	307
os-taillard-5	Average t	61.24	**91.43**	18.53	21.71	14.27
	# Solved	25/30	**30/30**	25/30	26/30	26/30
	# Nogoods			8275	12285	12285
os-taillard-7	Average t	97.61	18.31	1.69	59.09	**160.95**
	# Solved	7/30	8/30	8/30	9/30	**11/30**
	# Nogoods			912	14212	81264
os-taillard-10	Average t	147.00	86.60	1.21	1.14	**1.09**
	# Solved	4/30	**9/30**	**9/30**	**9/30**	**9/30**
	# Nogoods			581	581	581
os-taillard-15	Average t	60.71	14.59	29.28	20.74	**20.25**
	# Solved	6/30	12/30	**14/30**	**14/30**	**14/30**
	# Nogoods			700	700	700
os-taillard-20	Average t	0.79	115.74	121.78	183.34	**180.35**
	# Solved	1/30	15/30	16/30	**17/30**	**17/30**
	# Nogoods			34	150	150

Table 3 shows the results. Exploiting nogoods in restarts reduces the runtime and increases the number of solved instances in general. Using LightFilter, RST+NG$_{LG}$ solves the most number of instances with the best timing, except for problem set os-taillard-5. When the average number of nogoods of solved instances is small, the runtimes of RST+NG$_G$ and RST+NG$_{LG}$ are similar.

From Tables 3 and 4, we see that exploiting minimal/shortened nogoods allows more instances to be solved except with the last set os-taillard-20. The stronger pruning power of minimal nogoods than our shortened nogoods are demonstrated by the problem sets os-taillard-5 and os-taillard-7. However, the other harder problems benefit from shortened nogoods and our efficient global constraint filtering algorithms.

As explained earlier in the section, even though the pruning power of RST+NG and RST+NG$_{LG}$ are the same, their behaviors under dom/wdeg differ due to the essentially non-deterministic execution of filtering algorithms in Gecode (Schulte and Stuckey 2008). We thus evaluate our algorithms also with the classical deterministic dom/ddeg (Bessiere and Régin 1996) heuristic. Now RST+NG and RST+NG$_{LG}$ generate exactly the same search tree when solving the same problem. Their runtimes are now only affected by the relative runtime efficiency of the respectively filtering algorithms, which is the goal of this experiment. Since there are some pretty hard instances in the competition, we set a failure limit of $1,000,000$ to find a solution. Since RST+NG and RST+NG$_{LG}$ would have searched exactly the same space when a solution is found or the failure limit is reached when solving the same problem, we thus compare their average runtime by accounting both solved and unsolved instances for each set of problems. We also show the average number of nogoods to find a solution or reach the failure limit for each set.

Table 5 shows the results using dom/ddeg. Both the number of nogoods and the search tree size are the same

Table 4: Open Shop Scheduling using dom/wdeg with minimization and time limit

n		RST+NG^M	RST+NG^m_G	RST+NG$^m_{LG}$
os-taillard-4	Average t	0.04	**0.03**	**0.03**
	# Solved	**30/30**	**30/30**	**30/30**
	# Nogoods	101	98	98
os-taillard-5	Average t	**6.13**	21.45	6.51
	# Solved	**28/30**	**28/30**	**28/30**
	# Nogoods	2649	4095	2483
os-taillard-7	Average t	**48.20**	109.51	62.36
	# Solved	**14/30**	12/30	12/30
	# Nogoods	4375	4152	5148
os-taillard-10	Average t	76.50	75.78	**114.27**
	# Solved	15/30	17/30	**18/30**
	# Nogoods	2855	3530	4919
os-taillard-15	Average t	6.38	74.45	**74.05**
	# Solved	11/30	**16/30**	**16/30**
	# Nogoods	111	456	456
os-taillard-20	Average t	91.56	**144.05**	145.31
	# Solved	14/30	**16/30**	**16/30**
	# Nogoods	23	20	20

Table 5: Open Shop Scheduling using dom/ddeg without minimization/shortening

n		RST+NG	RST+NG$_G$	RST+NG$_{LG}$
os-taillard-4	Average t	56.56	32.99	**3.23**
	# Solved	**24/30**	**24/30**	**24/30**
	# Nogoods	13057	13057	13057
os-taillard-5	Average t	559.47	316.94	**35.27**
	# Solved	**3/30**	**3/30**	**3/30**
	# Nogoods	60110	60110	60110
os-taillard-7	Average t	1312.16	323.99	**110.96**
	# Solved	**2/30**	**2/30**	**2/30**
	# Nogoods	59333	59333	59333
os-taillard-10	Average t	3047.53	434.09	**298.08**
	# Solved	**2/30**	**2/30**	**2/30**
	# Nogoods	55057	55057	55057
os-taillard-15	Average t	2280.32	**612.99**	649.10
	# Solved	**9/30**	**9/30**	**9/30**
	# Nogoods	19471	19471	19471
os-taillard-20	Average t	2982.89	**2053.33**	2192.12
	# Solved	**13/30**	**13/30**	**13/30**
	# Nogoods	8778	8778	8778

for these three methods. RST+NG$_G$ runs 3.29 times faster than RST+NG on average. Using our light filtering algorithm, RST+NG$_{LG}$ runs 4.24 and 10.05 times faster than RST+NG$_G$ and RST+NG on average. RST+NG$_G$ performs the best in problem sets os-taillard-15 and os-taillard-20. The reason is, when a propagator is triggered, we not only prune values when necessary but also delete all satisfied positive and negative decisions to reduce the size of encoding sequences. While RST+NG$_G$ is triggered most often, it has the smallest overhead to record the encoding sequence during search. Now the overhead to record encoding sequence dominates the overhead of unnecessary triggers. Again, the advantage by exploiting minimal/shorter nogoods can be demonstrated in Table 6. More problems are solved and the overhead to find a solution is reduced. The number of nogoods is also substantially reduced. Again, RST+NG^m_G and RST+NG$^m_{LG}$ dominate in terms of timing.

Radio Link Frequency Assignment. Now we focus on the 12 hardest instances built from the real-world Radio Link Frequency Assignment Problem (RLFAP) which are also taken from the 2006 CSP Solver Competition[1]. Here, we set a time limit of 2 hours to find a solution.

Table 7 shows the results in a way similar to those for Open Shop Scheduling. More problems are solved after exploiting nogoods in restarts. RST+NG$_{LG}$ performs the best and is 1.37 times faster than RST+NG. After exploiting minimal or shortened nogoods in Table 8, the number of nogoods is reduced substantially. It takes up to 4% of runtime on average to do minimization for all solved problems. Shortening nogoods is even cheaper. Using FullFilter and shorter nogoods performs the best. However, comparing with Table 7, the performance is not improved utilizing minimal or shorter nogoods since these shorter nogoods do not seem to cooperate well with the dom/wdeg heuristic.

Conclusion

Our contributions are four fold. First, we demonstrate that reduced nld-nogoods extracted at a restart point are increasing. Second, since nogoods are not generated dynamically, we give a lightweight filtering algorithm which has the same consistency level with GAC on each nogood and is triggered less often. Third, minimizing nogoods destroy their increasing property, and we present an approximation to nogood minimization so that all shortened reduced nld-nogoods collected from the same restart are still increasing and thus our filtering algorithms can still be used. Fourth, we demonstrate the efficiency of our filtering algorithm and the approximate minimization procedure against existing state-of-the-art nogood recording techniques in restart-based search.

Experimental results of Open Shop Scheduling show that our shortened nogoods can sometimes cooperate well with the adaptive heuristic dom/wdeg. We also conduct extra experiments to take into account the weighted degree incurred by the nogoods filtering algorithm. Results show that ignoring the weighted degree performs better than not ignoring them in most of the cases. It is worthwhile to investigate the interaction of recorded nogoods with heuristics in restarts.

References

Bessiere, C., and Régin, J.-C. 1996. Mac and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *CP'96*, 61–75.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *ECAI'04*, 146–150.

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

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

Table 6: Open Shop Scheduling using dom/ddeg with minimization/shortening

n		RST+NG^M	RST+NG^m_G	RST+NG$^m_{LG}$
os-taillard-4	Average t	1.60	**1.00**	**1.00**
	# Solved	28/30	**29/30**	**29/30**
	# Nogoods	130	168	168
os-taillard-5	Average t	**37.49**	30.54	30.15
	# Solved	**7/30**	6/30	6/30
	# Nogoods	1106	1106	1106
os-taillard-7	Average t	93.07	77.59	**76.81**
	# Solved	**2/30**	**2/30**	**2/30**
	# Nogoods	1496	1304	1304
os-taillard-10	Average t	293.79	227.10	**225.06**
	# Solved	**2/30**	**2/30**	**2/30**
	# Nogoods	2004	2013	2013
os-taillard-15	Average t	739.49	**558.93**	625.76
	# Solved	**9/30**	**9/30**	**9/30**
	# Nogoods	1916	1945	1945
os-taillard-20	Average t	2317.54	**2117.61**	2154.00
	# Solved	**13/30**	**13/30**	**13/30**
	# Nogoods	1138	1136	1136

Table 7: RLFAP without minimization/shortening

RLFAP	NORST	RST	RST+NG	RST+NG$_G$	RST+NG$_{LG}$
Average t	35.15	780.35	1397.01	1041.88	**1017.33**
# Solved	5	6	**8**	**8**	**8**
# Nogoods			22606	25469	26114

Table 8: RLFAP with minimization/shortening

RLFAP	RST+NG^M	RST+NG^m_G	RST+NG$^m_{LG}$
Average t	464.82	**1382.09**	1588.86
# Solved	7	**8**	**8**
# Nogoods	1221	1840	2011

Gent, I. P.; Jefferson, C.; and Miguel, I. 2006. Watched literals for constraint propagation in Minion. In *CP'06*. Springer. 182–197.

Lecoutre, C.; Sais, L.; Tabary, S.; Vidal, V.; et al. 2007. Recording and minimizing nogoods from restarts. *JSAT* 1(3-4):147–167.

Lee, J., and Zhu, Z. 2014. An increasing-nogoods global constraint for symmetry breaking during search. In *CP'14*, 465–480.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47:173–180.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference*, 530–535. ACM.

Schiex, T., and Verfaillie, G. 1994. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal on Artificial Intelligence Tools* 3(2):187–207.

Schulte, C., and Stuckey, P. J. 2008. Efficient constraint propagation engines. *TOPLAS* 31(1):2:1–2:43.

Walsh, T. 1999. Search in a small world. In *IJCAI'99*, 1172–1177.