Combinaison de *nogoods* extraits au redémarrage

Gaël Glorian [*] **Frédéric Boussemart** **Jean-Marie Lagniez**
Christophe Lecoutre **Bertrand Mazure**

CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
{glorian, boussemart, lagniez, lecoutre, mazure}@cril.fr

Résumé

Dans cet article, nous nous intéressons à l'enregistrement de *nogoods*, instanciations partielles globalement incohérentes, pouvant être extraits systématiquement lors du redémarrage d'un algorithme complet (avec retour-arrière) de résolution CSP (problème de satisfaction de contraintes). Plus précisément, dans ce contexte, nous proposons plusieurs techniques de simplification et de combinaison de *nogoods*, dans le but d'accroître leur capacité de filtrage. La base de notre approche est une généralisation des *nld-nogoods* correspondant au concept introduit récemment d'*increasing-nogoods*. Nous proposons plusieurs algorithmes portant sur des combinaisons de sous-ensembles de *nogoods* identifiés de manière dynamique. Les similarités entre les différents *increasing-nogoods* permettent un meilleur élagage de l'arbre de recherche, notamment grâce à l'exploitation d'équivalences entre décisions. Nous proposons aussi quelques pistes d'amélioration, notamment un système de sentinelles et la génération de *nld-nogoods* à la volée. Nos résultats préliminaires montrent l'intérêt de notre approche pour certains problèmes.

Abstract

In this paper, we exploit nogoods, partial unsatisfiable instantiations, extracted from a restart-based search engine at the end of each run when the current cutoff value is reached. More precisely, in that context, we propose several simplification and combination techniques related to nogoods, in order to increase the filtering efficiency. The root of our approach is a generalization of nld-nogoods corresponding to the concept newly introduce of increasing-nogoods. We propose various algorithms relating to dynamically identified nogoods subsets combinations. Therefore, the search tree benefits from a better pruning thanks to similarities existing between *increasing-nogoods*, especially the equivalence between decisions. We also suggest some improvements tracks, in particular a sentinel system and on the fly nld-nogoods production. Our preliminary results show that our approach works well for several problems.

[*] Papier doctorant : Gaël Glorian est auteur principal.

1 Introduction

L'apprentissage de *nogoods* est un thème qui a été introduit dans les années 90 [2, 15, 4] pour la résolution de problèmes de satisfaction de contraintes. Les *nogoods* classiques, dits standards, sont des instanciations partielles ne pouvant mener à aucune solution. Ils ont été assez rapidement utilisés pour gérer l'explication [5, 7] de valeurs supprimées au cours de la recherche et de la propagation de contraintes. Ils ont ensuite été généralisés [8] en permettant la combinaison d'assignations de variables (décisions positives) et de réfutations de valeurs (décisions négatives). L'intérêt pratique des *nogoods* (généralisés) a été revisité par les travaux portant sur la génération paresseuse de clauses [3].

Les *nogoods* peuvent également être utiles dans un contexte de redémarrage régulier d'un algorithme de recherche arborescent. Il est en effet possible d'identifier [10] sur la dernière branche (celle qui est la plus à droite) un ensemble de *nogoods* représentant la partie de l'espace de recherche qui vient d'être explorée. Par le fait de simplement enregistrer ces *nogoods*, appelés *nld-nogoods* (réduits), on a la garantie de ne jamais explorer de nouveau les mêmes sous-arbres. Certaines extensions de ces travaux ont porté sur l'élimination de symétries [11, 13] et l'exploitation du caractère croissant des *nld-nogoods* [12], appelés de ce fait *increasing-nogoods*.

Dans cet article, nous proposons plusieurs techniques de simplification et de combinaison d'*increasing-nogoods*, nous permettant d'accroître leur capacité de filtrage. En analysant dynamiquement certaines combinaisons de sous-ensembles de *nogoods*, et en exploitant des formes d'équivalence entre décisions, nous montrons que l'arbre de recherche peut être mieux élagué. Nous identifions également quelques pistes prometteuses permettant de renforcer l'efficacité du processus de détection.

Cet article est structuré comme suit. Dans un premier temps, nous présentons les *nld-nogoods* ainsi que les *increasing-nogoods*. Puis, nous introduisons quelques techniques de combinaisons (avec leurs algorithmes) : combinaisons de décisions négatives, combinaisons par équivalence d'alpha, et combinaisons par équivalence de décisions négatives. Après avoir présenté quelques résultats expérimentaux, nous concluons et évoquons quelques pistes prometteuses.

2 Préliminaires

Un réseau fini de contraintes P est un couple $(\mathcal{X}, \mathcal{C})$, où $\mathcal{X}$ est un ensemble de variables et $\mathcal{C}$ un ensemble de contraintes. À chaque variable $x \in \mathcal{X}$ est associé un domaine, noté $dom(x)$ qui contient un ensemble fini de valeurs. Chaque contrainte $c \in \mathcal{C}$ porte sur un nombre fini de variables appelé la portée de c, noté $scp(c)$ et est définie formellement par une relation notée $rel(c)$ qui contient les tuples autorisés pour les variables de la portée, c'est-à-dire les combinaisons qui satisfont c. L'arité d'une contrainte c correspond au nombre de variables sur lesquelles la contrainte porte ($|scp(c)|$). Une solution de P est une instanciation de toutes les variables de P satisfaisant toutes les contraintes de P. Le problème CSP est un problème de décision qui consiste à déterminer si un réseau de contraintes donné admet une solution.

Un *nogood* est une instanciation partielle qui ne peut être prolongée vers une solution. L'intérêt des *nogoods* est d'éviter le phénomène de *thrashing*, c'est-à-dire d'explorer de manière répétée les mêmes sous-arbres incohérents. Deux approches classiques existent pour les identifier et les enregistrer : au cours de la recherche, ou au redémarrage. Dans cet article, nous allons nous placer dans le contexte d'un algorithme complet de résolution avec retour-arrière à branchement binaire et enregistrement de *nogoods* au redémarrage.

2.1 Nogood recording from restart

Les *nld-nogoods* [9] (*negative last decision nogoods*) sont extraits au redémarrage à partir de la dernière branche Σ de l'arbre de recherche. $\Sigma = \langle \delta_1, ..., \delta_m \rangle$ avec δ_i une décision positive, c'est-à-dire une assignation ($x = a$), ou négative, une réfutation ($x \neq a$). Ils représentent tous les conflits obtenus au cours du *run* précédent (i.e., depuis le dernier redémarrage). Une *nld-sous-séquence* est une séquence de décisions $\langle \delta_1, ..., \delta_j \rangle$ où δ_j est une décision négative. Pour toute nld-sous-séquence $\Sigma' = \langle \delta_1, ..., \delta_j \rangle$ de Σ, l'ensemble $\{\delta_1, ..., \neg\delta_j\}$ est un *nld-nogood* et l'ensemble $pos(\Sigma') \cup \{\neg\delta_j\}$, où $pos(\Sigma')$ correspond aux décisions positives de Σ', est un *nld-nogood* réduit. Il est connu que tous les *nld-nogoods* réduits extraits de la dernière branche de l'arbre de recherche subsument tous les *nld-nogoods* réduits qui pourraient être extraits des branches précédemment explorées.

Exemple. Considérons que l'algorithme soit sur le point d'effectuer un redémarrage, et que nous ayons l'arbre de recherche illustré par la figure 1.

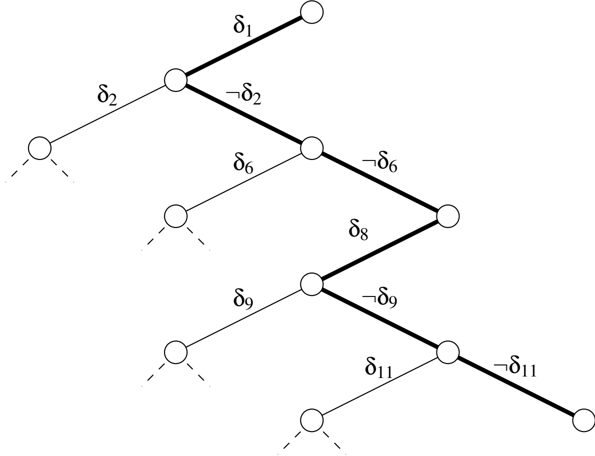

FIGURE 1 – Arbre de recherche avant redémarrage

La dernière branche de l'arbre de recherche est $\Sigma = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \neg\delta_{11}\}$. Nous pouvons en extraire les *nld-nogoods* suivants (à gauche ci-dessous) ainsi que les *nld-nogoods* réduits associés (à droite ci-dessous) :

$\Delta_1 = \{\delta_1, \delta_2\}$ $\quad\quad$ $\Delta_{1r} = \{\delta_1, \delta_2\}$
$\Delta_2 = \{\delta_1, \neg\delta_2, \delta_6\}$ $\quad\quad$ $\Delta_{2r} = \{\delta_1, \delta_6\}$
$\Delta_3 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \delta_9\}$ $\quad\quad$ $\Delta_{3r} = \{\delta_1, \delta_8, \delta_9\}$
$\Delta_4 = \{\delta_1, \neg\delta_2, \neg\delta_6, \delta_8, \neg\delta_9, \delta_{11}\}$ $\quad\quad$ $\Delta_{4r} = \{\delta_1, \delta_8, \delta_{11}\}$

Nous pouvons remarquer qu'il existe des similitudes entre les différents *nld-nogoods* extraits lors d'un même redémarrage, ils sont dits croissants. Les *increasing-nogoods* [12, 13] présentés dans la section suivante exploitent ce caractère croissant et permettent de représenter les *nld-nogoods* réduits extraits à la fin d'un *run* sous la forme d'une (seule) contrainte globale.

2.2 Increasing nogoods

Les *increasing-nogoods* (*IncNG*) [12, 13] sont une extension des *nld-nogoods* réduits. À chaque redémarrage une seule contrainte globale *IncNG* est ajoutée au réseau. Celle-ci représente tous les *nld-nogoods* réduits qui auraient pu être extraits lors de la recherche. L'idée est de mettre à disposition une structure compacte ainsi qu'un filtrage supérieur aux *nld-nogoods* réduits traités

indépendamment.

155 Pour passer d'un ensemble de *nld-nogoods* réduits à une contrainte *IncNG*, il faut commencer par les transformer en *nld-nogoods* réduits *dirigés*. Soit un *nld-nogood* réduit $\{\delta_i, \delta_j\}$, le *nld-nogood* réduit dirigé correspondant s'écrit $\delta_i \Rightarrow \neg\delta_j$.

Exemple. Revisitons l'arbre de recherche de l'exemple précédent en précisant les décisions (assignations et réfutations). Sur la branche $\Sigma = \langle\ x_2 = 1,\ x_3 \neq 0,\ x_4 \neq 1,\ x_5 = 2,\ x_1 \neq 1,\ x_6 \neq 2\ \rangle$ avec $dom(x_i) = \{0,1,2\}$, nous pouvons extraire l'ensemble suivant de *nld-nogoods* réduits :

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

qui sous forme dirigée s'écrivent :

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

Dans [12], les auteurs ont montré que l'ensemble des *nld-nogoods* réduits dirigés extraits d'une branche sont croissants (*increasing*), c'est-à-dire que $LHS(ng_i) \subseteq LHS(ng_{i+1})$ où *LHS* (*left hand side*) désigne la partie gauche de l'implication (similairement *RHS* désigne la partie droite).

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

Les *nld-nogoods* réduits dirigés ainsi obtenus peuvent s'écrire sous forme de séquence de décisions en éliminant les parties gauches redondantes :

$$\Sigma = \langle\ x_2 = 1,\ x_3 \neq 0,\ x_4 \neq 1,\ x_5 = 2,\ x_1 \neq 1,\ x_6 \neq 2\ \rangle$$

160 Nous remarquons que cela correspond à la séquence de décision initiale, celle extraite de l'arbre de recherche. Pour construire un *IncNG* il suffit donc de retenir la dernière branche avant le redémarrage.

165 Soit $\Lambda = \langle ng_0,\ \ldots, ng_t\rangle$ une séquence composée de *nld-nogoods* croissants. Si $LHS(ng_i)$ contient deux décisions positives pouvant encore être falsifiées alors les *nogoods* ng_j tel que $j \geq i$ sont nécessairement arc-cohérents (ou GAC pour Generalized Arc Consistency) car les parties

170 gauches des *nogoods* plus grands subsument celles des plus petits.

Pour filtrer la contrainte *IncNG*, deux indices α et β sont utilisés. Ils correspondent aux deux décisions positives

175 non assignées les plus à gauche dans la séquence (pouvant encore être falsifiées). Celles-ci sont surveillées ainsi que toutes les décisions négatives se situant entre α et β.

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

Il existe trois situations principales d'appel à l'algorithme de filtrage (algorithme 1) :

180
1. une décision négative contenue entre α et β est falsifiée, cela force δ_α à être faux, par conséquent tous les *nogoods* contenus dans la contrainte sont falsifiés ;
2. la décision positive désignée par α est satisfaite : nous forçons toutes les parties droites qui ne
185 contiennent que δ_α dans leur partie gauche à être vraies c'est-à-dire toutes les décisions négatives contenues entre α et β et nous recherchons la prochaine décision positive non assignée ;
3. la décision positive désignée par β est satisfaite : ceci
190 est semblable au cas précédent, nous recherchons la prochaine décision positive non assignée.

Algorithme 1 : FilterIncNG(Σ)

Data : $m = |\Sigma|$ where Σ is an *IncNG*
1 UpdateAlpha();
2 **if** $m \neq 0 \wedge \beta \neq m$ **then** UpdateBeta();
3 **if** $m = 0$ **then** delete constraint;

Les algorithmes 2 et 3 qui ont été proposés par [12] ont un fonctionnement relativement similaire. Si δ_α est

195 satisfait, un nouvel α est trouvé, l'algorithme est appelé de nouveau pour voir si δ_α est satisfaite jusqu'à arriver à un point fixe ou que la contrainte soit entièrement traitée (de la même manière pour β). La fonction *watchFollowDec* permet de trouver la prochaine décision positive à partir

200 de β si elle existe et de surveiller toutes les décisions négatives rencontrées. Si une décision négative falsifiée est rencontrée, m, qui représente la taille de l'*IncNG* traité, est mis à zéro, ce qui correspond à la désactivation de cet *IncNG*.

205

3 Combinaison d'*increasing-nogoods*

Les techniques existantes de traitement de *nogoods*, que ce soient les *nld-nogoods* ou les *increasing-nogoods*, se

Algorithme 2 : UpdateAlpha()

```
1  if δα is satisfied then
2      unsubscribe δα;
3      for i ∈ [α+1, β) do
4          if Neg(δi) then // true if δi is negative
5              satisfy δi;
6              unsubscribe δi;
7          end
8      end
9      if β = m then m ← 0; return ;
10     α ← β;
11     watchFollowDec();
12     if m ≠ 0 then UpdateAlpha();
13 else
14     for i ∈ [α+1, β) do
15         if Neg(δi) ∧ δi is falsified then
16             falsify δα;
17             m ← 0;
18             return ;
19         end
20     end
21 end
```

Algorithme 3 : UpdateBeta()

```
1 if δβ is satisfied then
2     unsubscribe δβ;
3     watchFollowDec();
4     if m ≠ 0 then UpdateBeta();
5 end
```

limitent à elles-mêmes. Ces méthodes ne tirent pas partie des autres informations disponibles, à savoir l'état des variables (c'est-à-dire, leur domaine) ainsi que les autres *nogoods*. Nous proposons donc dans cette section trois méthodes utilisant à bon escient ces informations. La première permet de continuer à traiter les *increasing-nogoods* indépendamment mais les fait toutefois interagir sur la base des domaines des variables qui les composent. Les deux autres regroupent les *increasing-nogoods* par sous-ensembles, soit en fonction de leurs décisions positives (δ_α), soit en fonction de leurs décisions négatives (celles qui sont comprises entre alpha et beta).

3.1 Combinaison de décisions négatives

Le but de la combinaison de décisions négatives est de sécuriser le fait qu'une assignation ou une suppression ne cause pas de conflit direct avec la base de *nogoods*, c'est-à-dire avant que l'*IncNG* relatif à la combinaison entre en conflit. Cela permet de déceler les conflits en amont et par conséquent d'augmenter le pouvoir de filtrage des *increasing-nogoods*.

Exemple. Soit le *IncNG* suivant :

$$ng_i = \langle\ x_2 = 1, x_3 \neq 2, x_3 \neq 4,\ x_5 = 3\rangle$$

Supposons qu'il soit dans son état initial, c'est-à-dire $\alpha \leftarrow x_2 = 1$ et $\beta \leftarrow x_5 = 3$ et que les variables aient toutes le même domaine $dom(x_i) = \{1,2,3,4\}$. Si x_2 est assigné à 1, alors il est possible de supprimer les valeurs 2 et 4 du domaine de x_3 (voir la situation 2 de l'algorithme 1). Si le domaine de x_3 ne contenait plus que ces deux valeurs alors il y a conflit. Ce conflit aurait pu être détecté avant, voire évité, si la valeur 1 avait été supprimée du domaine de x_2 lorsque le domaine de x_3 a été réduit à $\{2,4\}$.

Par souci de simplicité, nous considérons que pour *ng*, un *IncNG* donné, les valeurs de alpha et beta sont accessibles par $\alpha(ng)$ et $\beta(ng)$. Nous pouvons alors définir `diffValues`(ng,x), la fonction qui retourne les valeurs impliquées dans une décision négative de *ng* impliquant *x* et située entre $\alpha(ng)$ et $\beta(ng)$. De même, `diffVars`(ng) est la fonction qui retourne l'ensemble des variables impliquées dans une décision négative de *ng* située entre $\alpha(ng)$ et $\beta(ng)$. Par exemple, si nous considérons l'*IncNG* $ng_{ex} = \langle\ x_2 = 1, x_3 \neq 2, x_3 \neq 4,\ x_5 = 3\rangle$ avec $\alpha(ng_{ex}) = x_2$ et $\beta(ng_{ex}) = x_5$, alors `diffVars`$(ng_{ex}) = \{x_3\}$ et `diffValues`$(ng_{ex}, x_3) = \{2,4\}$.

Algorithme 4 : checkNegativeDecisions(ng)

```
  Data : Let ng be an IncNG
1 foreach x ∈ diffVars(ng) do
2     if diffValues(ng,x) ⊇ dom(x) then
3         falsify α(ng);
4     end
5 end
```

L'algorithme 4 réalise ce filtrage qui consiste donc à effectuer une inférence (réfuter la valeur impliquée dans $\alpha(ng)$) chaque fois qu'un conflit est susceptible de se produire. Notre approche, bien que se limitant à l'analyse des *increasing-nogoods* de manière indépendante, offre une capacité de filtrage renforcée. Dans la section suivante, nous proposons une nouvelle variation de ce principe par analyse de la base complète d'*increasing-nogoods*. L'algorithme 4 a une complexité dans le pire cas en $O(sn)$ où *s* est la taille du plus grand *IncNG* (majoré par *dn* où *d* est la taille du plus grand domaine) et *n* le nombre de variables.

3.2 Combinaison par équivalence d'alpha

Dans cette partie, nous étendons le principe présenté précédemment à des ensembles d'*increasing-nogoods*. Pour cela, nous partitionnons l'ensemble des *increasing-nogoods* en fonction des valeurs des δ_α : les *increasing-nogoods* de même valeur se situent dans le même groupe. Il faut donc maintenir ces groupes de manière dynamique, c'est-à-dire les mettre à jour à chaque modification d'un alpha, lors d'un filtrage ou d'un retour-arrière. En raisonnant avec ces groupes, nous obtenons une capacité de filtrage renforcée (englobant le cas précédent).

Exemple. Soit $dom(x_i) = \{0,1,2,3\}$,

$$IncNG_0 \equiv \quad ..., x_6 \neq 2, \underbrace{x_2 = 1}_{\alpha}, x_1 \neq 3, \underline{x_3 \neq 1}, ...$$

$$IncNG_1 \equiv \quad ..., x_2 \neq 0, x_1 \neq 2, \underbrace{x_2 = 1}_{\alpha}, \underline{x_3 \neq 0}, ...$$

$$IncNG_2 \equiv \quad ..., \underbrace{x_2 = 1}_{\alpha}, \underline{x_3 \neq 2}, x_6 \neq 1, x_8 \neq 3, ...$$

Sur l'exemple, nous pouvons constater que $x_2 = 1$ est le δ_α commun à un groupe de trois *increasing-nogoods*. En regardant les décisions négatives qui suivent ces trois occurrences de δ_α (la valeur de beta n'est pas importante pour notre illustration), nous remarquons que trois valeurs différentes pour x_3 apparaissent dans les décisions négatives surveillées (avant le beta qui n'est pas représenté). Cela signifie que si x_2 venait à être assigné à 1 la seule valeur restante dans le domaine de x_3 serait 3. De ce fait, si la valeur 3 a déjà été retirée du domaine de x_3, il faut alors absolument empêcher x_2 de pouvoir être assignée à 1. Pour empêcher cela, tous les *increasing-nogoods* du groupe vont être désactivés (car forcées à être toujours vérifiés après avoir supprimé la valeur 1 du domaine de x_2). L'algorithme 5 réalise ce filtrage.

Algorithme 5 : checkNegativeDecGroup(`ngGroup`)

Data : Let `ngGroup` be a set of *increasing-nogoods* with a common δ_α

1. `X` = $\bigcup \{x_i \in$ `diffVars`$(ng_j) \mid ng_j \in$ `ngGroup`$\}$;
2. **foreach** $x \in$ `X` **do**
3.   `V`=$\bigcup\{v_i \in$`diffValues`$(ng_j, x) \mid ng_j \in$`ngGroup`$\}$;
4.   **if** `V` $\supseteq dom(x)$ **then**
5.   falsify α(`ngGroup`);
6.   **end**
7. **end**

L'algorithme 5 crée en premier lieu l'ensemble `X` qui contient toutes les variables apparaissant entre alpha et beta du groupe passé en paramètre de l'algorithme. De plus, il constitue pour chaque variable contenue dans `X` un ensemble de valeurs `V`. L'ensemble de valeurs `V` ainsi constitué est comparé au domaine de la variable relative, si ceux-ci sont équivalents, la décision pointée par alpha du groupe *ngGroup* est falsifié en vue d'éviter un conflit possible dans la suite de la recherche. L'algorithme 5 a une complexité dans le pire cas en $O(nsg)$ où n le nombre de variables total, s est la taille du plus grand *IncNG* et g le nombre d'*increasing-nogoods* dans la base.

Nous allons voir qu'il est possible de traiter les *increasing-nogoods* ayant la même variable pointée par alpha mais avec des valeurs différentes et ainsi extraire des sous-groupes ayant des décisions négatives communes de ces ensembles.

3.3 Combinaison par équivalence de décisions négatives

De la même manière que précédemment, les *increasing-nogoods* sont ici aussi traités par ensemble. La différence réside dans le fait que nous regroupons ceux qui ont un alpha de variable commune et surveillent une même décision négative (que nous appelons pivot) située entre les alphas et betas courants. Si toutes les valeurs restantes dans le domaine de la variable commune des alphas sont présentes dans le groupe, alors la décision négative pivot doit être vérifiée.

Exemple. Soit $\forall i \in \mathbb{N},\ dom(x_i) = \{0,1,2,3\}$,

$$IncNG_0 \equiv \quad ..., x_6 \neq 2, \underbrace{x_2 = 1}_{\alpha}, x_1 \neq 3, \underline{x_3 \neq 1}, ...$$

$$IncNG_1 \equiv \quad ..., x_7 \neq 0, x_1 \neq 2, \underbrace{x_2 = 0}_{\alpha}, \underline{x_3 \neq 1}, ...$$

$$IncNG_2 \equiv \quad ..., \underbrace{x_2 = 2}_{\alpha}, \underline{x_3 \neq 1}, x_6 \neq 1, x_8 \neq 3, ...$$

$$IncNG_3 \equiv \quad ..., x_4 \neq 3, \underbrace{x_2 = 3}_{\alpha}, \underline{x_3 \neq 1}, x_1 \neq 1, ...$$

Sur l'exemple nous pouvons voir que $x_3 \neq 1$ est une décision négative surveillée commune à l'ensemble d'*increasing-nogoods*. En regardant les alphas nous remarquons que les quatre valeurs possibles pour x_2 apparaissent. C'est-à-dire que peu importe la valeur que prend x_2 alors x_3 est toujours différent de 1 à ce stade de la recherche.

En considérons cela, nous pouvons minimiser indirectement les *increasing-nogoods* selon les scénarios de recherche, voire directement si toutes les valeurs de l'alpha

d'un groupe apparaissent.

Algorithme 6 : checkAlphaAndNegDec()

1 **foreach** $x \in$ `alphaSet` **do**
2   **if** `diffValuesAlpha`$(x) \supseteq$ *dom(x)* **then**
3         $\Delta = \bigcap\{\delta_i \in ng_j | j \in$ `alphaToNG(x)`$\}$;
4   **foreach** $\delta \in \Delta$ **do**
5   satisfy δ;
6   **end**
7   **end**
8 **end**

Soit `alphaSet`, un ensemble dynamique des différentes variables pointées par alpha dans chaque *IncNG*, l'algorithme 6 permet de rechercher les décisions négatives communes dans les groupes qui portent sur la même variable pointée par alpha (la valeur peut être différente). Dans ce but, nous utilisons deux fonctions :

— `diffValuesAlpha(x)` qui permet de retourner l'ensemble de valeurs différentes prises pour une même décision x pointée par alpha dans la base globale d'*increasing-nogoods*,
— `alphaToNG` qui utilise un ensemble de correspondances entre la variable de l'alpha et les *increasing-nogoods* où elle apparaît.

Ces fonctions permettent de construire l'ensemble Δ qui contient les décisions négatives communes à un groupe où toutes les valeurs d'une même variable pointée par alpha apparaissent. L'algorithme 6 a une complexité dans le pire cas en $O(ndg^2)$ où n est le nombre de variables total, d la taille du plus grand domaine et g le nombre d'*increasing-nogoods* dans la base.

4 Expérimentations

Les expériences présentées ci-après sont réalisées sur des machines équipées d'un processeur *Intel Xeon X5550* cadencé à 2,67 GHz ainsi que de 8 Go de mémoire, avec un *timeout* réglé à 15 minutes. Nous avons sélectionné un panel représentatif de 3744 problèmes provenant des compétitions CP de 2006 et de 2008. Les résultats sont présentés sous forme de tableaux où les familles sont regroupées.

Pour ces expérimentations, nous avons utilisé une version simplifiée du solveur *rclCSP* présenté dans [6] utilisant l'heuristique *dom/wdeg* [1]. Étant donnée la complexité de l'algorithme 6 et le fait que l'algorithme 4 soit subsumé par l'algorithme 5, nous avons choisi de ne reporter que ce dernier. Dans la suite, il est référencé par le symbole $\alpha_{\Leftrightarrow}$ dans les tableaux.

Afin de montrer la corrélation entre le nombre et la taille des *nogoods*, nous utilisons différentes stratégies de redémarrage basées sur le nombre de conflits. Celles-ci sont, soit basées sur la suite de Luby [14] (1,1,2,1,1,2,4,. . .), soit sur une suite géométrique. Pour ces expérimentations, nous avons utilisé les politiques suivantes :

— P50 - suite géométrique de premier terme 50 et de raison 1.5 ;
— P10 - suite géométrique de premier terme 10 et de raison 1.1 ;
— L100 - suite de luby dont les termes sont multipliés par 100.

Le tableau 1 reporte une comparaison entre les *increasing-nogoods* avec et sans $\alpha_{\Leftrightarrow}$ sur la famille de problème *QCP-20*. Dans ce tableau, nous reportons le nombre total de *nld-nogoods* apparaissant dans les *increasing-nogoods* (*#nld*), le nombre total de suppressions induites par les *increasing-nogoods* (*#del*), en parenthèse apparaît le nombre de suppressions dues à $\alpha_{\Leftrightarrow}$ (ce nombre est inclus dans le total). Ce tableau reporte aussi le nombre de conflits imputés par la révision des *increasing-nogoods* (*#fail*), ainsi que le temps de résolution, exprimé en secondes (*cpu*). Sur ce tableau, nous pouvons remarquer que sur cette famille d'instances l'ajout d'$\alpha_{\Leftrightarrow}$ permet de résoudre une instance de plus. Nous pouvons aussi remarquer que de nombreuses suppressions de valeurs sont induites par l'ajout de notre méthode. Finalement, nous observons que pour la majorité des problèmes de cette famille, le nombre de conflits obtenus dans les *increasing-nogoods* est plus faible. Cela peut être s'expliquer par le fait que les conflits sont détectés plus tôt et donc que l'arbre de recherche est plus petit.

Le tableau 2 permet d'évaluer l'impact de la stratégie de redémarrage sur les performances des méthodes considérées. Pour cela nous avons comparé trois méthodes, *increasing-nogoods* avec et sans $\alpha_{\Leftrightarrow}$ ainsi que les *nld-nogoods*. Cette table reporte, pour l'ensemble des problèmes considérés dans cette expérience, le nom de la famille considérée (*Family name*) ainsi que le nombre de problèmes dans celle-ci (*#prob*). Pour chacune des méthodes, nous reportons la stratégie de redémarrage considérée ainsi que le nombre d'instances résolues (*#sol*) et le temps moyen de résolution (*cpu*). Comme nous pouvons voir sur ce tableau, une stratégie de redémarrage moins agressive est souhaitable lorsque nous considérons les *nld-nogoods* ou les *increasing-nogoods* seuls. Dans le cas de notre méthode $\alpha_{\Leftrightarrow}$, une stratégie plus agressive permet d'utiliser pleinement la puissance des *increasing-nogoods*. Cela peut en partie s'expliquer par le fait d'une base plus importante de *nogoods* permet d'induire un

nombre plus important de suppressions.

Lors de nos expérimentations nous avons remarqué que plus nous utilisons une politique de redémarrage agressive plus les combinaisons sont efficaces d'un point de vue suppressions mais la complexité [1], dépendant de la taille de la base, augmente. Nous avons donc lancé quelques simulations pour mesurer l'impact en temps de calcul sur une politique relativement agressive, à savoir *luby* × *10*. Nous avons empêché les algorithmes relatifs aux *increasing-nogoods* ainsi qu'à $\alpha_{\Leftrightarrow}$ de faire le moindre changement sur le réseau mais ils continuent à être appelés normalement. Le résultat est le suivant sur l'instance scen11-f5 le temps de résolution est de 120 secondes dont 85 de calcul d'équivalence d'alpha et 2,5 secondes de gestion des *increasing-nogoods*. Cette même instance est résolue en 17,45 secondes (dont 3,9 secondes de combinaisons) si nous activons les suppressions. De ces résultats, nous envisageons quelques pistes d'amélioration dans la section suivante.

5 Travaux futurs

À la vue des résultats encourageants nous avons décidé d'améliorer les algorithmes présentés précédemment grâce à diverses méthodes. Dans cette section nous allons présenter une première méthode, à savoir un système de sentinelles.

Les sentinelles peuvent s'apparenter à un système de *1-watch* appliqué à des domaines relatifs à des *increasing-nogoods*. Nous allons pouvoir les utiliser dans deux des trois méthodes de combinaison présentées, la combinaison de décisions négatives (présentée section 3.1) ainsi que la combinaison par équivalence de décisions négatives (présentée section 3.3).

Dans le cas de la combinaison de décisions négatives, la théorie est la suivante : dans les décisions négatives contenues entre alpha et beta par un *IncNG*, c'est-à-dire là où nous espérons avoir des valeurs supprimées, nous pouvons ajouter un certain nombre de sentinelles. Une sentinelle marque une valeur que nous ne regardons pas normalement dans le domaine de la variable associée. Tant que celle-ci existe nos suppressions ne créeront pas de conflits. De plus, grâce à ce système de sûreté de domaine nous n'avons plus à nous soucier des valeurs négatives lors du filtrage d'un *IncNG* dans le cas où δ_α n'est pas assigné, nous pouvons, par conséquent, supprimer les lignes 13 à 21 de l'algorithme 2.

Nous proposons de mettre en place un système de sentinelles qui, associé à chaque *IncNG*, regarde si une valeur différente de celle apparaissant dans le *IncNG* existe. Tant qu'il reste un élément autre que ceux apparaissant dans le *IncNG*, les suppressions pourront se faire sans conflit si δ_α venait à être satisfaite. Si la sentinelle venait à être supprimée sans possibilité d'en trouver une nouvelle, il faudrait alors supprimer la valeur pointée par α. Cela peut se faire en utilisant soit des *1-watch*, soit de manière paresseuse en vérifiant la taille des domaines concernés par rapport aux nombres de valeurs qui apparaissent négativement dans le *IncNG*, cette dernière étant beaucoup moins précise.

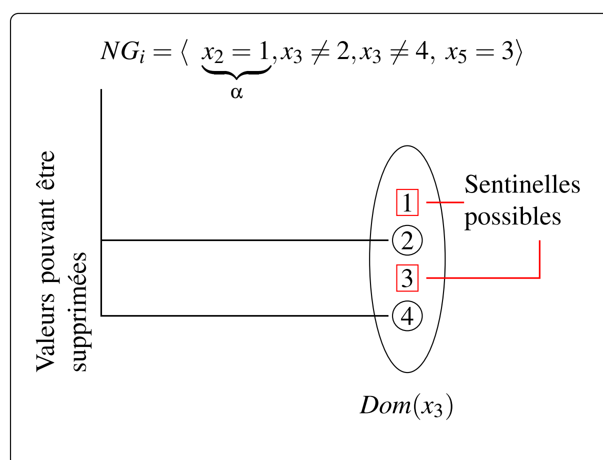

FIGURE 2 – Exemple de fonctionnement des sentinelles

Nous pouvons voir, dans la figure 2, qu'il est possible de choisir deux sentinelles, les valeurs 1 et 3 du domaine de x_3. Tant qu'au moins une de ces valeurs est dans le domaine, nous n'avons pas besoin de falsifier la décision pointée par alpha.

Dans le cas de la combinaison par équivalence de décisions négatives, nous pouvons utiliser une sentinelle pour détecter lorsque la taille du domaine de l'alpha commun (même variable, valeur différente) arrive à la taille du sous-groupe.

Toujours dans le but d'étendre les interactions entre *nogoods* nous proposons une méthode alternative, voire complémentaire, aux combinaisons proposées dans cet article. Nous considérons toujours des groupes avec décisions négatives équivalentes mais avec des alphas quelconques. Grâce à ceux-ci, nous générons des *nld-nogoods* réduits composés des prémices, c'est-à-dire les décisions positives avant l'alpha courant, ainsi que des alphas. Cette méthode peut se généraliser aux combinaisons par équivalence de décisions négatives ainsi qu'aux combinaisons par équivalence d'alpha.

1. $O(nsg)$ où n est le nombre de variables, s la taille du plus grand *IncNG* et g la taille de la base d'*IncNG*.

P10	IncNG + $\alpha_{\Leftrightarrow}$				IncNG			
	cpu	#nld	#del (dont $\alpha_{\Leftrightarrow}$)	#fail	cpu	#nld	#del	#fail
qcp-20-187-0	222,61	671	38	1	**208,05**	671	38	1
qcp-20-187-1	176,41	806	11033(708)	289	**31,75**	569	2287	107
qcp-20-187-2	timeout	1019	57232(2088)	6665	timeout	1007	54728	6980
qcp-20-187-3	timeout	770	33255(2027)	1881	timeout	844	53057	5043
qcp-20-187-4	timeout	818	39867(2548)	1905	timeout	862	47013	3692
qcp-20-187-5	**0,7**	121	15	0	**0,7**	121	15	0
qcp-20-187-6	634,87	835	5177(1)	573	**597,85**	835	5177	573
qcp-20-187-7	**258,52**	679	1267(17)	40	timeout	847	1582	90
qcp-20-187-8	**32,94**	462	807(116)	6	205,33	667	5286	350
qcp-20-187-9	**15,54**	396	72	2	15,62	396	72	2
qcp-20-187-10	timeout	921	21082(667)	1362	timeout	892	43964	3780
qcp-20-187-11	**0,27**	63	44(22)	3	0,3	62	38	8
qcp-20-187-12	timeout	880	21178(540)	1524	timeout	926	34894	6362
qcp-20-187-13	timeout	863	102101(2906)	8127	timeout	946	191009	33687
qcp-20-187-14	timeout	983	137251(13389)	6261	timeout	930	30693	1819

TABLE 1 – Résultats expérimentaux détaillés sur quelques problèmes.

6 Conclusion

Cet article a mis en évidence qu'il était possible de combiner les informations entre les *increasing-nogoods* et la structure du problème afin d'augmenter significativement le pouvoir de filtrage. Plus précisément, nous avons d'abord proposé une approche qui, étant donné un *nogood*, prend en considération les informations relatives aux domaines des variables afin d'identifier de nouvelles valeurs à supprimer. Ensuite, nous avons montré qu'il est possible d'étendre ce processus à la base entière d'*increasing-nogoods* de deux manières, soit en fonction de leurs décisions positives, soit en fonction de leurs décisions négatives.

Nous avons montré sur un large panel d'instances que ces algorithmes de filtrage permettent de supprimer de nombreuses valeurs lors du processus de propagation. Cependant, l'utilisation de ces algorithmes a deux inconvénients. Le premier est que le fait de détecter les conflits en amont à un impact sur l'heuristique de choix de variables. En effet, puisque l'heuristique utilisée est ajustée en fonction des conflits, les éviter ne permet pas d'augmenter la pondération des contraintes, et donc de choisir la meilleure variable. L'autre inconvénient est la complexité des algorithmes présentés, qui dépend de la taille de la base d'*increasing-nogoods*. Comme énoncé dans la section 5 cette situation peut être améliorée en considérant des mécanismes de mise à jour basés sur l'utilisation d'un système de sentinelles ou d'ensembles persistant *backtrack-ready*.

Références

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

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

[3] T. Feydy and P. Stuckey. Lazy clause generation reengineered. In *Proceedings of CP'09*, pages 352–366, 2009.

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

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

[6] E. Gregoire, J.M. Lagniez, and B. Mazure. A CSP solver focusing on fac variables. In *Proceedings of CP'11*, pages 493–507, 2011.

[7] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings of CP'00*, pages 249–261, 2000.

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

[9] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. In *Proceedings of IJCAI'07*, pages 131–136, 2007.

570 [10] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1 :147–167, 2007.

[11] C. Lecoutre and S. Tabary. Symmetry-reinforced nogood recording from restarts. In *Proceedings of SymCon'11*, pages 13–27, Perugia, Italy, 2011.
575

[12] J. H. M. Lee, C. Schulte, and Z. Zhu. Increasing nogoods in restart-based search. In *Proceedings of AAAI'16*, pages 3426–3433, 2016.

580 [13] J. H. M. Lee and Z. Zhu. An increasing-nogoods global constraint for symmetry breaking during search. In *Proceedings of CP'14*, pages 465–480, 2014.

[14] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Inf. Process. Lett.*, 47(4) :173–180, 1993.
585

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

		NLD						IncNG						IncNG + $\alpha_{\Leftrightarrow}$					
		L100		P50		P10		L100		P50		P10		L100		P50		P10	
Family name	#prob	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu	#sol	cpu
allIntervalSeries	25	10	15,77	**12**	**45,12**	11	15,48	10	13,45	11	7,31	11	19,39	10	13,93	11	7,34	11	35,81
bddLarge	35	35	60,3	35	52,26	35	128,88	**35**	**30,49**	35	30,54	35	33,15	35	30,56	35	30,74	35	33,16
bddSmall	35	35	109,52	35	43,35	35	113,69	35	25,4	35	25,2	35	25,65	35	25,3	**35**	**24,99**	35	25,48
bibd10-11	6	3	3,88	3	232,16	1	0,21	2	0,85	**4**	**175,44**	2	2,43	2	1,27	4	178,82	2	2,31
bibd12-13	7	**5**	10,09	3	1,78	4	1,97	4	4,87	4	10,63	3	5,29	4	4,39	4	11,36	3	5,12
bibd8	7	3	166,18	3	95,17	3	58,05	3	88,95	3	61,82	**3**	41,97	3	180,02	3	63,5	3	44,52
bibd9	10	10	1,46	10	2,47	10	4,53	10	1,41	10	2,41	8	4,3	**10**	**1,37**	10	2,2	8	4,31
bibdVariousK	29	23	7,22	23	60,36	22	28,68	**25**	**31,27**	21	4,44	23	6,47	24	12,78	22	42,04	23	7,53
bmc	24	24	8,19	24	14,32	24	30,47	24	7,99	24	15,11	24	22,57	**24**	**7,94**	24	16,06	24	20,82
chessboardColoration	20	15	6,76	15	7,82	15	21,81	15	3,66	15	5,67	15	6,4	**15**	**3,65**	15	5,67	15	5,76
coloring	22	21	0,58	22	4,37	22	8,51	22	9,44	**22**	**3,48**	22	3,89	22	16,64	22	4,28	22	4,85
compet08	16	9	85,22	9	107,12	9	132,69	8	70,17	**10**	**151,98**	9	81,72	9	157,09	10	157,34	9	87,2
costasArray	11	7	52,19	9	78,21	9	133,82	8	104,29	8	71,13	**9**	**62,42**	8	96,51	8	68,77	9	64,41
dag-half	25	1	898,31	6	631,48	0	NaN	17	412,64	16	370,99	17	385,25	**18**	**408,44**	16	406,4	17	373,83
dag-rand	25	24	448,24	25	163,81	25	282,27	25	136,01	25	132,93	25	131,57	25	134,59	25	132,27	**25**	**129,97**
dubois	13	0	NaN	6	275,72	4	261,5	4	246,52	**6**	**155,64**	6	174,93	3	319,69	6	166,99	6	214,52
fapp11-15	55	41	80,75	41	41,49	42	46,44	43	62,34	42	52,14	42	128,43	**43**	**61,48**	42	53,05	42	37,93
frb30-15	10	10	0,91	**10**	**0,68**	10	1,02	10	0,87	10	0,78	10	0,84	10	0,84	10	0,83	10	0,87
frb35-17	10	10	5,22	10	2,75	10	3,13	10	1,95	**10**	**1,81**	10	1,97	10	2,24	10	1,91	10	2,05
frb40-19	10	10	210,05	10	31,11	10	39,64	10	33,4	10	29,61	10	29,57	10	33,87	**10**	**27,25**	10	29,02
frb45-21	10	1	398,13	10	248,76	9	265,95	9	222,56	9	229,33	**10**	**233,17**	9	243,82	9	251,02	10	283,14
frb50-23	10	0	NaN	**2**	**457,53**	1	456,36	0	NaN	1	844,67	1	794,87	1	454,4	1	850,63	1	858,43
golombRulerArity3	14	9	57,76	11	131,92	10	61,76	10	50,99	10	51,37	10	51,36	10	56,17	**11**	**123,79**	10	49,83
golombRulerArity4	14	8	78,86	**10**	**55,28**	10	79,38	10	62,02	10	58,74	10	59,17	10	62,57	10	58,96	10	59,15
graphColoring	458	185	19,15	196	25,86	196	30,84	196	27,81	**198**	**32,11**	197	27,81	195	30,54	197	28,66	197	28,68
lexHerald	47	39	33,22	41	39,32	41	41,51	**44**	**55,01**	41	66,24	43	59,36	44	69,71	40	33,47	43	63,28
lexVg	63	62	50,3	63	32,6	63	44,45	63	30,74	**63**	**27,55**	63	30,74	63	29,04	63	28,38	63	30,58
magicSquare	18	5	3,01	7	114,26	6	6,51	6	103,33	6	59,44	7	89,88	6	105,37	6	59,93	**7**	**89,02**
marc	10	**10**	**8,1**	10	8,11	10	8,15	10	8,5	10	8,23	10	8,35	10	8,41	10	8,32	10	8,22
mknap	6	4	78,03	4	39,8	4	91,26	4	23,06	**4**	**22,67**	4	23,2	4	22,72	4	22,74	4	22,91
nengfa	10	9	123,78	8	59,26	9	79,37	9	85,85	9	89,78	**9**	**61,02**	9	89,71	9	79,23	9	83,71
ogdHerald	50	**50**	**2,4**	50	2,85	50	4,61	50	2,56	50	2,58	50	3,07	50	2,48	50	2,6	50	3,08
ogdPuzzle	22	22	0,76	22	0,76	22	0,78	22	0,84	22	0,83	22	0,71	22	0,79	**22**	**0,7**	22	0,75
ogdVg	65	34	73,9	39	93,04	35	81,07	**40**	**74,33**	39	55,27	39	63,24	40	77,1	40	81,11	40	81,09
ortholatin	9	3	31,98	3	15,46	**4**	**102,44**	3	54,38	3	13,77	3	34,57	3	54,27	3	13,7	3	33,37
os-taillard-10	30	20	33,42	17	42,09	20	31,87	20	32,34	18	65,09	**20**	**30,4**	20	32,69	18	64,08	20	30,68
os-taillard-15	30	21	7,49	20	7,22	21	9,39	21	6,94	20	6,68	21	8,84	21	7,01	**20**	**6,86**	21	8,86
os-taillard-20	30	22	24,23	21	26,03	22	29,06	22	22,18	21	24,66	22	28,69	**22**	**20,47**	21	24	22	28,65
os-taillard-4	30	30	0,78	30	0,67	30	0,79	30	0,59	30	0,69	30	4,07	30	0,69	**30**	**0,58**	30	0,7
os-taillard-5	30	28	49,5	29	39,24	29	47,16	29	37,04	28	28,33	**30**	**27,97**	30	57,02	29	57,61	30	28,44
os-taillard-7	30	16	72,55	17	72,95	**19**	**85,25**	16	65,69	17	54,9	18	133,96	17	73,57	17	55,04	18	134,68
pseudoGLB	384	132	310,84	187	135,54	152	296,18	**198**	**73,22**	190	66,81	186	67,01	189	72,04	189	64,38	186	77,17
pseudo	486	257	36,88	268	29,1	258	35,12	**281**	**36,13**	272	30,34	274	26,8	278	32,09	272	31,46	274	30,61
QCP-15	15	15	9,2	15	2,85	15	4,79	15	2,04	15	2,57	**15**	**1,77**	15	2,35	15	2,61	15	2,31
QCP-20	15	5	137,33	7	49,31	6	93,76	8	208,02	7	133,62	7	151,93	6	55,57	7	164,33	**8**	**168,18**
QG3	7	5	2,12	5	1,71	5	2,45	5	1,46	**5**	**1,15**	5	1,7	5	1,6	5	1,33	5	1,67
QG4	7	5	2,78	5	2	5	2,88	5	1,95	**5**	**1,53**	5	2,36	5	1,58	5	1,54	5	2,31
QG5	7	6	75,46	6	15,97	6	22,21	6	13,24	6	12,6	**6**	**12,52**	6	13,96	6	12,69	6	12,62
QG6	7	6	6,97	6	3,28	6	6,04	6	2,06	6	2,12	6	1,12	6	2,09	6	1,97	**6**	**1,1**
QG7	7	4	15,62	5	7,37	5	11,91	**5**	**4,7**	5	53,16	5	11,92	5	4,76	5	54,03	5	12,2
queenAttacking	10	5	25,4	5	4,11	5	2,42	5	2,63	5	50,82	5	3,99	**5**	**2,37**	5	50,75	5	3,74
queensKnights	18	16	70,54	17	99,89	17	91,62	16	57,53	**18**	**105,63**	18	116,45	16	58,46	18	105,68	18	130,59
QWH-20	10	9	50,02	10	101,23	10	80,04	10	88,01	**10**	**40,99**	10	48,26	9	22,38	10	68,25	10	49,01
QWH-25	10	0	NaN	1	590,09	1	875,89	1	794,82	**3**	**310,66**	0	NaN	1	838,62	2	265,2	2	458,58
rand-10-20-10	20	20	2,78	20	2,81	20	2,97	**20**	**2,64**	20	2,84	20	2,94	20	2,66	20	2,82	20	2,89
rand-2-23	10	10	103,09	10	33,17	10	38,01	10	29,65	**10**	**26,72**	10	32,53	10	36,07	10	29,35	10	28,17
rand-2-24	10	10	405,85	10	82,45	10	79,37	10	81,45	10	69,53	**10**	**67,9**	10	98,83	10	71,09	10	71,03
rand-2-25	10	3	515,25	10	155,91	10	173,81	10	180,25	10	141,47	10	149,84	10	280,6	**10**	**125,12**	10	138,37
rand-2-26	10	1	55,54	10	372,33	10	397,49	10	401,18	10	339,67	10	346,05	5	161,03	**10**	**320,91**	10	392,58
rand-2-27	10	0	NaN	3	357,2	4	275,15	4	435,21	4	295,76	**4**	**236,98**	3	295,63	4	459,59	4	341,94
rand-2-30-15-fcd	50	50	0,83	50	0,72	50	0,88	50	0,74	**50**	**0,7**	50	0,77	50	0,75	50	0,79	50	0,8
rand-2-30-15	50	50	1,13	50	1	50	1,06	**50**	**0,94**	50	0,98	50	1	50	1,04	50	0,99	50	1
rand-2-40-19-fcd	50	48	194,81	50	26,88	50	41,12	50	29,1	**50**	**25,52**	50	29,75	50	32,4	50	31,95	50	29,92
rand-2-40-19	50	40	296,87	50	66,82	50	81,37	50	63,86	50	61,55	**50**	**61,36**	50	71,01	50	64,06	50	62,17
rand-2-50-23-fcd	50	0	NaN	14	505,07	5	462,74	12	524,45	15	457,11	12	470,7	9	381,09	15	522,33	**15**	**456,83**
rand-2-50-23	50	0	NaN	12	514,2	4	685,69	12	473,67	12	597,81	9	539,14	8	519,13	**13**	**568,28**	9	460,74
rand-3-20-20-fcd	50	50	56,15	50	28,07	50	34,29	50	28,55	**50**	**22,67**	50	26,77	50	30,42	50	23,41	50	24,69
rand-3-20-20	50	50	104,04	50	51,46	50	56,07	50	47,14	50	45,94	50	47,82	50	48,9	**50**	**44,71**	50	47,33
rand-3-24-24-fcd	50	10	413,14	26	272,73	23	326,17	28	314,43	26	297,64	27	314,23	**29**	**272,14**	26	271,52	28	301,76
rand-3-24-24	50	6	536,52	16	467,63	14	415,31	15	425,36	14	367,36	16	401,12	16	405,91	15	478,29	**18**	**390,44**
rlfapScens11	12	10	84,9	11	103,52	11	112,19	11	98,85	11	101,48	11	103,31	10	39,88	**11**	**98,48**	11	112,5
schurrLemma	10	8	65,17	8	76,64	8	92,92	9	71,41	9	158,85	8	77,14	**9**	**71,1**	8	78,43	9	120,94
super-jobShop	46	33	12,35	35	31,83	34	15,24	36	36,12	35	24,63	33	6,68	**36**	**26,86**	34	20,62	35	45,39
tdsp	42	5	30,47	5	38,76	5	171,76	5	2,12	5	73,06	5	5,43	5	1,71	6	103,54	**6**	**27,42**
tightness0,1	100	100	11,33	100	4,6	100	6,13	100	4,42	**100**	**4,07**	100	4,28	100	4,54	100	4,17	100	4,62
tightness0,2	100	100	17,92	100	6,49	100	8,67	100	6,07	**100**	**5,7**	100	6,17	100	6,57	100	6,01	100	6,18
tightness0,35	100	100	13,27	100	6,18	100	7,18	100	5,53	**100**	**5,33**	100	5,59	100	5,88	100	5,44	100	5,68
tightness0,5	100	100	16,68	100	7,29	100	8,21	100	6,87	100	7,19	100	7,1	100	7,24	**100**	**6,86**	100	7
tightness0,65	100	100	7,23	100	5,05	100	5,41	**100**	**4,55**	100	4,74	100	4,63	100	4,86	100	4,82	100	4,66
tightness0,8	100	100	6,27	100	4,67	100	5,35	100	5,01	**100**	**4,64**	100	4,75	100	4,68	100	4,69	100	4,78
tightness0,9	100	100	7,83	100	6,86	100	7,08	100	7,12	100	7,4	100	7,35	100	7,11	100	7,47	**100**	**6,66**
TOTAL	3744	2443		2648		2571		2687		2668		2665		2662		2669		2679	

TABLE 2 – Résultats expérimentaux par familles des trois méthodes : NLD, IncNG et IncNG + $\alpha_{\Leftrightarrow}$ sur différentes politiques de redémarrage.