Hybridation de prouveurs CSP et apprentissage

Julien Vion

Hybridation de prouveurs CSP et apprentissage

Julien Vion

CRIL-CNRS FRE 2499,
Université d'Artois
Lens, France
vion@cril.univ-artois.fr

Résumé

À ce jour, l'algorithme MGAC-$dom/wdeg$, qui maintient l'Arc Consistance Généralisée pendant la recherche d'une solution, est considéré comme étant l'approche générique la plus efficace pour résoudre des Problèmes de Satisfaction de Contraintes (CSP) difficiles et de grande taille. Dans cet article, nous proposons une approche hybride capable de combiner des recherches systématiques et locales indépendantes tout en transférant des informations utiles d'un algorithme à l'autre. Nous proposons différentes interactions, et en particulier l'apprentissage de nogoods, la pondération de contraintes ainsi que des affectations gloutonnes. Sur un grand nombre d'instances de CSP structurés, les résultats expérimentaux montrent que notre approche donne des résultats intéressants en comparaison avec MGAC-$dom/wdeg$.

Abstract

Today, the MGAC-$dom/wdeg$ algorithm, i.e., the algorithm which maintains Generalized Arc Consistency during the search of a solution, is still considered as the most efficient generic approach to cope with large and hard problem instances of the Constraint Satisfaction Problem (CSP). In this paper, an hybrid approach combining tree search with local search is proposed. Several combination schemes are considered, combining nogood learning, constraint weighting, Tabu lists, restarts and greedy assignments. On many CSP instances, experimental results of our approach show interesting improvements on the well known MGAC-$dom/wdeg$ algorithm.

1 Introduction

De nombreux problèmes d'intelligence artificielle et d'informatique gravitent autour la satisfaction de contraintes. On rencontre ces problèmes en ordonnancement, planification, biologie moléculaire ou encore dans la conception de circuits. Les problèmes de contraintes sont généralement représentés par des réseaux de contraintes (*Constraint Networks* ou *CN*). Un CN P est composé d'un ensemble fini de variables $\mathscr{X}$ et d'un ensemble fini de contraintes $\mathscr{C}$. Pour trouver une solution à un CN, il faut chercher une valeur pour chaque variable de telle sorte que chaque contrainte soit satisfaite. Le Problème de Satisfaction de Contraintes (*Constraint Satisfaction Problem* ou *CSP*) consiste à déterminer si un CN (également appelé *instance de CSP*) est consistant, c'est à dire si il admet au moins une solution. Cet article concerne les CSP discrets, c'est à dire les CSP dont les variables admettent un nombre fini de valeurs, généralement un sous-ensemble de $\mathbb{Z}$.

Les méthodes de recherche développées pour la résolution de CSP appartiennent généralement à une des deux grandes familles : algorithmes systématiques et algorithmes de recherche locale. À l'heure actuelle, on utilise généralement l'algorithme MGAC pour résoudre une instance de CSP. MGAC est un algorithme de recherche systématique arborescent, qui maintient l'Arc Consistance Généralisée *GAC* au cours de la recherche. GAC est une propriété importante des CN, assurant que toutes les valeurs ont au moins un support dans chacune des contraintes du problème. On part du CSP original, et on affecte une valeur à chaque variable une par une jusqu'à l'obtention d'une solution. Un mécanisme de retour-arrière (*backtrack*) permet l'exploration de toutes les hypothèses pouvant mener à toutes les solutions du problème. Le principal inconvénient de ces algorithmes basés sur le retour-arrière, c'est que si une mauvaise hypothèse est faite au départ, il faudra explorer un sous-arbre de taille potentiellement très importante sans succès. En établissant l'Arc Consistance Généralisée à chaque nœud de l'arbre, on traite en partie ce problème en détectant à l'avance les hypothèses qui ne mèneront à aucune solution [19]. Une autre possibilité est d'utiliser des heuristiques de choix de variable permettant de se focaliser

sur les parties les plus difficiles du CN, ce qui permet également de réduire le nombre d'hypothèses à effectuer avant d'effectuer des retour-arrières. $dom/wdeg$ est à ce jour une des heuristiques de choix de variables généralistes les plus efficaces [2].

Les algorithmes de recherche locale comme *Hill-Climbing Min-Conflicts* [16] ou la *Recherche Tabu* [6] partent d'une affectation aléatoire complète de l'ensemble des variables, puis explorent de manière incomplète l'espace de recherche en *réparant* cette affectation. Bien que ces algorithmes ne tirent généralement pas profit de l'inférence comme GAC, ils sont souvent beaucoup plus efficaces que les algorithmes systématiques pour trouver une solution. Cependant, les algorithmes de recherche locale sont incomplets, c'est à dire qu'ils ne fournissent aucune garantie de trouver une solution ou de prouver l'incohérence d'un problème.

L'idée de combiner les avantages de la recherche systématique et de la recherche locale en concevant des algorithmes hybrides n'est pas neuve, et de nombreux travaux ont étudié la coopération entre la recherche locale et la recherche systématique, au point de devenir un véritable défi pour les problèmes de satisfaction de contraintes [21]. Dans cet article, nous présentons une approche hybride consistant à alterner l'exécution de deux algorithmes, un de chaque type. De plus, nous proposons de faire interagir ces deux algorithmes entre eux et d'établir une coopération basée sur l'apprentissage et la communication d'informations d'une exécution à l'autre.

Nous présentons un puissant algorithme de recherche systématique, MGAC-$dom/wdeg$, ainsi qu'un algorithme efficace de recherche locale, WMC. Nous verrons comment faire interagir les deux algorithmes dans notre approche hybride et comparons celle-ci avec d'autres travaux similaires. Enfin, nous présentons des résultats expérimentaux encourageants.

2 Préliminaires

2.1 Les Problèmes de Satisfaction de Contraintes

Définition 1. Un Réseau de Contraintes (*Constraint Network* ou CN) est un couple $(\mathscr{X}, \mathscr{C})$ où :

- $\mathscr{X}$ est un ensemble fini de n variables telles qu'à chaque variable X est associé un domaine $dom(X)$, indiquant l'ensemble de valeurs autorisées pour X,
- $\mathscr{C}$ est un ensemble fini de e contraintes telles qu'à chaque contrainte C est associée une relation $rel(C)$ indiquant l'ensemble des couples de valeurs autorisées pour les variables $vars(C) \subseteq \mathscr{X}$ impliquées par la contrainte C.

Dans le présent article, d représentera la taille du plus grand domaine et r l'arité maximale des contraintes. Le degré d'une variable correspond au nombre de contraintes impliquant celle-ci. Le degré maximal d'une variable dans un CN sera noté Γ_{max}. La valeur $a \in dom(X)$ sera notée X_a.

Une solution à un CN est une affectation de valeurs à toues les variables de sorte que toutes les contraintes soient satisfaites. Un CN est dit cohérent si et seulement si il admet au moins une solution. Le Problème de Satisfaction de Contraintes (CSP) consiste à déterminer si un réseau de contraintes est cohérent. Il s'agit d'un problème NP-complet. Une instance de CSP est définie par un réseau de contraintes, et la résoudre consiste à trouver au moins une solution ou à déterminer son incohérence. Pour résoudre une instance de CSP, il est possible de modifier le réseau de contraintes en utilisant des méthodes d'inférence ou de recherche. En général, on utilise des méthodes d'inférence pour filtrer des valeurs inconsistantes, ne pouvant apparaître dans aucune solution [4]. La principale méthode d'inférence reste l'Arc Consistance Généralisée (GAC).

L'Arc Consistance Généralisée garantit l'existence d'un support pour chaque valeur dans chaque contrainte. Établir cette propriété sur un CN P consiste à retirer chaque valeur qui n'est pas arc-consistante. Une valeur $v \in dom(X)$ est arc-consistante si dans toutes les contraintes impliquant X il existe au moins un support pour v. L'Arc Consistance (AC) est simplement GAC limité aux CN binaires (chaque contrainte implique exactement deux variables).

2.2 L'algorithme MGAC

L'algorithme MGAC [19] permet de résoudre une instance de CSP en effectuant une recherche en profondeur d'abord avec retour-arrière, tout en maintenant GAC. Plus précisément, à chaque nœud de la recherche, une affectation est effectuée, suivie d'un processus de filtrage nommé propagation de contraintes, qui correspond à l'établissement de GAC. Quand cet algorithme est appliqué aux instances de CSP binaires, il est nommé simplement MAC.

Les récentes implantations de MGAC utilisent un système de branchement binaire [10] : à chaque nœud de la recherche, une variable X est sélectionnée, une valeur $a \in dom(X)$ est choisie, et deux branches sont considérées : la première correspond à l'affectation de la valeur à la variable ($X = a$), l'autre à la suppression de cette valeur du domaine de la variable ($X \neq a$).

L'Algorithme 1 correspond à une version récursive de l'algorithme MGAC (utilisant le branchement binaire). Une instance de CSP est résolue par l'appel de $MGAC$, qui renvoie **vrai** si et seulement si l'instance est consistante. $P|_{X=a}$ représente le CN obtenu de P en réduisant le domaine de X au singleton $\{a\}$. $P|_{X\neq a}$ représente le CN obtenu de P en enlevant a du domaine de X. $P \backslash X$ représente le CN obtenu de P en retirant la variable X.

Les heuristiques permettant la sélection de X_a ont été

Algorithme 1 : MGAC(P = $(\mathscr{X}, \mathscr{C})$: CN, $maxBT$: Entier) : Booleen

```
1 si maxBT < 0 alors lever Expiration
2 si X = ∅ alors retourner vrai
3 sélectionner X_a tel que X ∈ X et a ∈ dom(x)
4 P' ← GAC(P|_{X=a})
5 si P' ≠ ⊥ ∧ MGAC(P'\X, maxBT) alors
6     retourner vrai
7 P' ← GAC(P|_{X≠a})
8 retourner P' ≠ ⊥ ∧ MGAC(P', maxBT − 1)
```

identifiées comme étant un point crucial pour l'efficacité des algorithmes de recherche. En utilisant différentes heuristique de choix de variables, on peut obtenir des résultats très différents en terme d'efficacité de la recherche.

Lecoutre et al. proposent dans [2] d'associer un compteur, noté $wght[C]$, à chaque contrainte C du problème. Ces compteurs sont utilisés pour pondérer les contraintes. Quand on constate l'incohérence d'une contrainte pendant la phase de propagation, son poids est incrémenté.

Le degré pondéré d'une variable X est alors défini comme la somme des poids de toutes les contraintes impliquant X et au moins une autre variable non affectée. L'heuristique adaptative $dom/wdeg$ [2] consiste à sélectionner d'abord la variable présentant le plus petit ratio "taille du domaine courant" par "degré pondéré courant". Lors de la progression de la recherche, le poids des contraintes difficiles devient de plus en plus important, ce qui aide grandement l'heuristique à sélectionner les variables apparaissant dans les parties les plus difficiles du CN. De nombreux travaux montrent l'efficacité pratique de cette heuristique [2, 12, 9, 23, 3].

2.3 L'algorithme de recherche locale WMC

La recherche locale a déjà été envisagée pour résoudre les CSP. Contrairement aux algorithmes de recherche systématique comme MGAC, les techniques de recherche locale sont par nature incomplètes : si une solution existe, il n'existe aucune garantie de la trouver en temps fini, et l'absence de solution ne peut être prouvée. Cependant, sur les instances de très grande taille, la recherche locale reste en pratique la meilleure alternative.

Un algorithme de recherche locale fonctionne sur des *affectations complètes* : une valeur est affectée à chaque variable, puis l'affectation est itérativement *réparée* jusqu'à la découverte d'une solution. Une réparation implique en général le changement d'une valeur affectée à une variable, de telle sorte que le moins de contraintes possibles soient violées [16].

Il arrive fréquemment qu'aucune réparation simple ne puisse améliorer l'affectation courante en termes de satisfaction de contraintes. Dans ce cas, un *minimum local* a été atteint. Le principal défi de la recherche locale consiste à trouver le meilleur moyen de sortir ou d'éviter les minima locaux afin de poursuivre la recherche. Diverses techniques ont été étudiées dans de précédents travaux :

Mouvements aléatoires Avec une probabilité p, une réparation est effectuée aléatoirement au lieu d'être choisie parmi les meilleures réparations possibles. Le premier algorithme utilisant cette technique est décrite dans [16] et sera appelée par la suite MCRW *Min-Conflicts with Random Walks* dans cet article.

Liste Tabu Les réparations précédentes (un couple variable, valeur) sont enregistrées afin d'éviter les réparations pouvant mener à des affectations déjà visitées [6]. Un nombre limité de réparations (correspondant à la taille de la liste Tabu) est gardé en mémoire, et les plus anciens sont oubliés, de sorte qu'un grand nombre de réparations reste disponible à chaque itération. Le *critère d'aspiration* permet de choisir une réparation présente dans la liste Tabu si celle-ci permet d'obtenir une affectation meilleure que toutes celles rencontrées jusqu'à présent.

Redémarrages Une limite au nombre d'itérations est paramétrée, à la manière du paramètre $maxFlip$ dans GSAT [22] (une itération consiste soit à réparer l'affectation courante ou à pondérer les contraintes dans un minimum local), de sorte que l'algorithme puisse être exécuté plusieurs fois sur des affectations initiales différentes.

Ces trois techniques nécessitent la mise au point d'importants paramètres, respectivement p, la taille de la liste Tabu, et un critère d'expiration. Nous nous proposons d'utiliser une autre technique pour échapper aux minima locaux non paramétrée, tout en conservant les redémarrages, qui nous permettront de mettre en œuvre une forme d'hybridation d'algorithmes.

Pondération de Contraintes La pondération des contraintes, inspirée de la *Breakout Method* de Morris [17] constitue le point central de l'algorithme que nous avons sélectionné. Elle permet un échange d'informations très naturel avec MGAC-$dom/wdeg$. Quand un minimum local est rencontré, toutes les contraintes non satisfaites sont pondérées.

L'algorithme de recherche locale par pondération de contraintes, WMC (*Weighted Min-Conflicts*), est décrit par l'Algorithme 4. La structure de données $\gamma(X, v)$, dont la gestion est décrite par les Algorithmes 2 et 3 permet de maintenir dynamiquement et de manière incrémentale le nombre de conflits qu'amènera chaque réparation. Comme

Algorithme 2 : initγ(P = $(\mathscr{X}, \mathscr{C})$: CN)

```
1  foreach X ∈ 𝒳 do
2      foreach v ∈ dom(X) do
3          γ(X, v) ← 0
4          foreach C ∈ 𝒞 | X ∈ vars(C) do
5              if ¬check(C|_{X=v}) then
6                  γ(X, v) ← γ(X, v) + wght[C]
```

Algorithme 3 : updateγ(X : Variable, v_{old} : Value)

```
1  foreach C ∈ 𝒞 | X ∈ vars(C) do
2      foreach Y ∈ vars(C) | X ≠ Y do
3          foreach v_y ∈ dom(Y) do
4              if check(C|_{Y=v_y}) ≠
               check(C|_{Y=v_y ∧ X=v_old}) then
5                  if check(C|_{Y=v_y}) then
6                      γ(Y, v_y) ← γ(Y, v_y) − wght[C]
7                  else
8                      γ(Y, v_y) ← γ(Y, v_y) + wght[C]
```

chaque affectation n'a un impact que sur les contraintes impliquant la variable sélectionnée, il est possible d'effectuer la sélection du meilleur couple (variable, valeur) à réparer en $O[(n + \Gamma_{max})rd]$.

Démonstration. Soit $P(\mathscr{X}, \mathscr{C})$ un réseau de contraintes et $\mathscr{A}$ l'affectation courante de toutes les variables $\in \mathscr{X}$. Soit $\gamma(X, v)$ une structure de données devant contenir le nombre de conflits de chaque voisin de A (différant de A par une réparation correspondant à l'affectation d'une valeur v à une variable X).

Étant donné $\mathscr{A}$, un nombre supposé connu de contraintes sont en conflit (qui peut être obtenu en $O(e)$ lors d'une première phase d'initialisation). Tous les voisins de A consistent en une affectation différente d'une variable X. Seules les contraintes impliquant X doivent donc être contrôlées. Le nombre de voisins de A est borné par $O(nd)$ et le nombre de contraintes impliquant chaque variable est borné par $O(\Gamma_{max})$. L'initialisation de $\gamma(X, v)$ peut donc être faite en $O(\Gamma_{max}nd)$.

Après la sélection d'un voisin $\mathscr{A}'$, $\gamma(X, v)$ doit être mis à jour. Cependant, $\mathscr{A}'$ ne diffère à son tour de $\mathscr{A}$ que par une seule variable X. Les contrôles effectués lors de l'étape précédente peuvent donc être conservés, et seules les contraintes impliquant X doivent être considérées. Il faut alors prendre en compte toutes les variables impliquées par chaque contrainte (borné par $O(r)$). Lors de toutes les étapes suivantes, il faut donc effectuer $O(\Gamma_{max}rd)$ contrôles pour mettre à jour $\gamma(X, v)$ avant de choisir la meilleure affectation en $O(nd)$. Le coût total d'une itération est donc de $O[(n + \Gamma_{max})rd]$. □

Algorithme 4 : WMC(P = $(\mathscr{X}, \mathscr{C})$: CN, $maxIterations$: Entier) : Booleen

```
1  pour chaque X ∈ 𝒳 faire
2      sélect X_a t.q. countWC(P|_{X=a}) minimal
3      P ← P|_{X=a}
4  nbConflicts ← countWC(P)
5  initγ(P)
6  nbIterations ← 0
7  tant que nbConflicts > 0 faire
8      sélect X_v | γ(X, v) minimal
9      v_old ← valeur actuelle de X
10     si γ(X, v) ≥ γ(X, v_old) alors
11         pour chaque C ∈ 𝒞 | C est en conflit faire
12             wght[C]++ ; nbConflicts++
13             pour chaque Y ∈ vars(C) faire
14                 pour chaque
                   w ∈ dom(Y) | ¬check(C|_{Y=w}) faire
15                     γ(Y, w)++
16     sinon
17         P ← P|_{X=v}
18         nbConflicts ← γ(X, v)
19         updateγ(P, v_old)
20     si nbIterations++ > maxIterations alors
21         lever Expiration
22 retourner vrai
```

Le coût en espace de $\gamma(X, v)$ est évidemment en $O(nd)$. L'utilisation d'une telle structure de données a été proposé par [6] et est utilisable également avec MCRW et la recherche Tabu.

Les lignes 1-6 de l'Algorithme 4 effectuent l'initialisation. $countWC(P)$ renvoie le nombre pondéré de contraintes $C \in \mathscr{C}$ violées. Les lignes 7 à 21 effectuent la recherche locale proprement dite. La ligne 10 détecte les minima locaux. Quand un minimum local est rencontré, le poids de toutes les contraintes en conflit est incrémenté (ligne 12), puis $\gamma(X, v)$ est mise à jour en $O(erd)$.

3 Recherches alternées

3.1 Une hybridation séquentielle simple

La manière la plus simple d'hybrider deux algorithmes reste de les lancer en parallèle, et de s'arrêter dès qu'un des deux renvoie une solution. Comme il est possible d'utiliser un système de redémarrage avec la plupart des algorithmes (en utilisant un critère d'expiration), nous proposons ici

de les lancer en *séquence*. Dans les deux principes d'hybridation, les algorithmes restent indépendants. Ainsi, les avantages des deux stratégies se combinent : s'il existe une meilleure stratégie pour un problème donné (par exemple, la recherche locale pour les problèmes denses de grande taille, ou la recherche systématique pour les problèmes incohérents), un temps limité sera perdu avec les mauvaises stratégies de recherche.

De plus, grâce à notre principe séquentiel, il est possible d'extraire de nombreuses informations intéressantes d'une exécution à l'autre, même si la recherche échoue, d'une manière similaire à la technique décrite dans [15], dont nous discuterons ci-dessous. Nos travaux se focalisent sur l'alternance de l'algorithme systématique MGAC-$dom/wdeg$ avec WMC. Nous montrons comment des informations peuvent être transmises d'un algorithme à l'autre, et comment la combinaison de ces algorithmes mène à une amélioration globale du processus de recherche sur les CSP structurés.

3.2 À propos de la pondération de contraintes

Le principal défaut des techniques de recherche systématique comme MGAC est bien connu : de mauvaises hypothèses au début de la recherche peuvent conduire à explorer de très grands sous-arbres qui pourraient être évités si l'heuristique avait au se focaliser sur de petits sous-problèmes très difficiles ou même incohérents. On dit alors que le prouveur est soumis au *"trashing"* : il redécouvre les mêmes incohérences un grand nombre de fois. D'autre part, on peut constater que certaines instances ne sont en réalité pas très difficiles à résoudre (par exemple, l'arbre minimal pour prouver leur incohérence est très réduit), mais peuvent avoir un comportement *"heavy tailed"* si elles sont mélangées aléatoirement et résolues plusieurs fois.

La pondération de contraintes utilisée par $dom/wdeg$ a été conçue pour identifier les sous-problèmes difficiles et permet de limiter grandement les phénomènes de *"trashing"* et *"heavy-tail"* sur les problèmes structurés [2].

Mazure et al. [15] indiquent que les statistiques sur les contraintes violées pendant la recherche locale peuvent être utilisées efficacement comme oracle pour une recherche systématique. On peut donc penser que le poids des contraintes obtenu après l'exécution de WMC se révèlera très utile pour initialiser les poids de $dom/wdeg$. La Figure 1 montre l'efficacité pratique de cette hypothèse. Le graphe représente la répartition des poids des contraintes après une simple exécution de WMC sur un gros problème incohérent d'allocation de fréquences radio (RLFAP). Après 50 000 itérations, les 28 contraintes connues pour appartenir à au moins un noyau minimalement inconsistant (mises en évidence sur la figure) obtiennent un poids moyen de 2 590 alors que le poids moyen sur l'ensemble

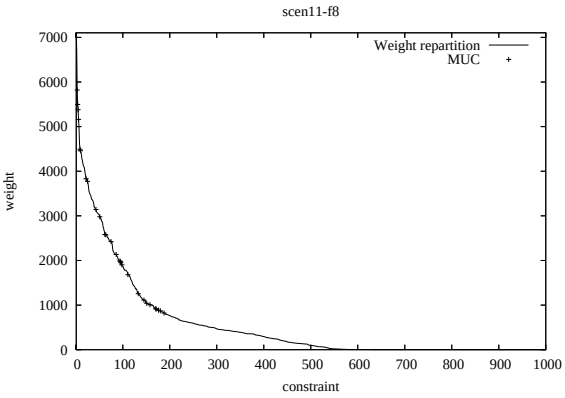

FIG. 1 – Répartition des poids après une exécution de WMC sur scen11-f8

des 4 103 contraintes est de 136.

3.3 Utiliser une recherche systématique partielle pour extraire des nogoods

Le principal avantage de la recherche systématique réside dans sa capacité à prouver l'incohérence des problèmes. En utilisant une recherche binaire comme décrit en section 2.2, l'heuristique effectue une hypothèse de type $X = a$ à chaque chaque nœud, ce qui crée deux sous problèmes : $GAC(P|_{X=a})$ et $GAC(P|_{X\neq a})$. Les deux sous-problèmes sont explorés en séquence jusqu'à ce qu'une solution soit trouvée ou que l'incohérence des deux sous-problèmes soit prouvée.

Afin d'améliorer les performances de la recherche et éviter le problème des mauvaises hypothèses, l'utilisation d'une stratégie de redémarrage combinée avec des heuristiques adaptatives s'est révélée très efficace pour résoudre les CSP avec MGAC [8]. Un nombre maximal de retour-arrières est fixé, et la recherche interrompue et reprise du début avec de nouvelles hypothèses. Cependant, cela signifie que toute la recherche effectuée avant le redémarrage est perdue. Lecoutre et al. proposent d'utiliser des nogoods afin de garder une trace des sous-problèmes inconsistants est éviter de les explorer encore et encore [14].

Quand la recherche expire, des nogoods sont identifiés à partir de données extraites d'une partie limitée (de taille linéaire) de la trace de l'arbre de recherche, et ajoutés au réseau de contraintes sous la forme de nouvelles contraintes. Tous les nogoods sont des conséquences logiques du problème et peuvent être considérés comme des contraintes additionnelles et redondantes qui enregistrent le travail effectué par les algorithmes. Pour des raisons pratiques, la taille des nogoods est limitée de sorte que le graphe de contraintes ne devienne pas trop dense. Plusieurs nogoods ayant la même portée sont fusionnés naturellement pour former une contrainte unique.

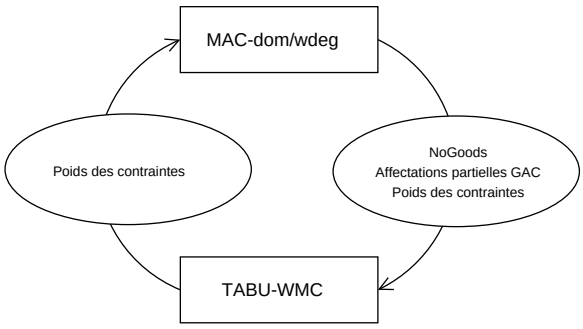

FIG. 2 – Le système d'hybridation

Le nombre de nogoods stockés à l'expiration d'une recherche est borné par le nombre d'hypothèses *encore en cours* au moment où la recherche est interrompue. Le nombre d'hypothèses positives ($X = a$) est bornée par le nombre de variables n. La taille des nogoods stockés dépend directement du nombre d'hypothèses positives. Pour ne pas créer de contraintes d'arité trop importante, difficiles à gérer, on limite la taille des nogoods à λ (en général, on choisit $\lambda = r$). Le nombre d'hypothèses négatives est borné par nd. Le nombre de nogoods appris à l'interruption de chaque exécution est donc dans le pire des cas de $O(\lambda nd)$. *On a la garantie que pour un arbre de recherche partiel donné, il ne peut y avoir de nogoods subsumés par (i.e. consistant une conséquence logique de) un autre nogood. Le nombre de nogoods est donc minimal.*

Les prochaines exécutions de WMC et MGAC-$dom/wdeg$ pourront toutes deux utiliser ces contraintes additionnelles pour guider la recherche ou filtrer l'espace de recherche plus efficacement. Cependant, il faut relativiser l'impact de ces nogoods, puisque l'apprentissage d'un nogood de taille λ implique l'exploration d'une fraction potentiellement importante ($\frac{1}{d^\lambda}$) de l'espace de recherche.

3.4 Algorithme hybride

Une information supplémentaire peut être conservée depuis MGAC-$dom/wdeg$ vers la recherche locale : quand MGAC-$dom/wdeg$ expire, l'affectation partielle courante, ainsi que tout le réseau de contraintes résultant est arc-consistant. Il est possible de générer l'affectation complète initiale pour WMC à partir de ce réseau.

Toutes les variables affectées conservent leurs valeurs. Les autres variables sont affectées de manière gloutonne : une valeur est sélectionnée dans les valeurs arc-consistantes restantes, de sorte que aussi peu de contraintes que possible soient violées. WMC cherche alors à réparer cette affectation. Une récapitulation de l'algorithme final est présentée Figure 2.

Bien que WMC et MGAC-$dom/wdeg$ ne nécessitent pas d'autres paramètres que $maxIterations$ et $maxBT$, respectivement, trouver les bonnes valeurs fut extrêmement

Algorithme 5 : Hybrid(P = $(\mathscr{X}, \mathscr{C})$: CN, $maxIter$: Entier) : Booleen

```
1  maxTries = 1
2  maxBT = maxIter × 8n/ed
3  répéter
4      startTime ← now()
5      répéter ⌊maxTries⌋ fois
6          essayer
7              retourner WMC(P, maxIter)
8          capturer Expiration
9      WMCDuration ← now() − startTime
10     startTime ← now()
11     essayer
12         retourner MGAC(P, maxBT)
13     capturer Expiration
14     MGACDuration ← now() − startTime
15     maxTries ← α × maxTries
16     maxBT ← α × maxBT × WMCDuration/MGACDuration
```

délicat, le paramètre idéal dépendant encore une fois fortement du problème à traiter. Une quantité limitée de ressources doit être allouée à chaque algorithme. Nous avons donc cherché à faire en sorte que chaque algorithme utilise autant de ressources l'un que l'autre.

La durée d'une exécution de WMC peut être contrôlée en fixant un nombre maximal d'itérations ($maxIterations$ dans l'Algorithme 5). La durée d'une itération dépend fortement de la taille du problème. Notre implantation Java effectue asymptotiquement environ $\frac{25\ 000\ 000}{nd}$ itérations par seconde sur des CSP aléatoires binaires avec un processeur i686 cadencé à 3 GHz. Notez que pour des problèmes industriels comme les RLFAP (Radio Link Frequency Assignment Problem) ou FAPP (Frequency Assignment Problem with Polarization constraints), nd atteint facilement 20 000 et dépasse les 2 000 000 sur les plus grosses instances (qui sont cependant facilement résolues par MGAC-$dom/wdeg$ de par la nature hautement hétérogène de ces problèmes). Notre implantation Java de MGAC-$dom/wdeg$ effectue asymptotiquement environ $\frac{200\ 000\ 000}{ed^2}$ hypothèses par seconde sur des problèmes binaires aléatoires.

À chaque itération de l'algorithme hybride principal, $maxTries$ exécutions de WMC sont effectuées (lignes 5-8), puis MGAC-$dom/wdeg$ est exécuté une fois (lignes 11-13). À chaque fois, $maxTries$ et $maxBT$ sont augmentés par un facteur α, de sorte que MGAC-$dom/wdeg$ puisse être complet. Dans le pire des cas, ceci surviendra quand $maxBT \geq d^n$. $maxBT$ est ajusté dynamiquement de sorte que les ressources allouées à la recherche locale et à la recherche systématique soient identiques.

4 Discussion

L'idée d'hybrider une recherche locale avec une recherche systématique n'est pas nouvelle, et plusieurs travaux étudient la coopération entre les deux types de recherche [11, 20, 5, 18, 15].

On peut catégoriser les approches en trois grandes catégories :

- effectuer une recherche locale avant ou après une recherche systématique
- effectuer une recherche systématique augmentée d'une recherche locale à un point de la recherche : à chaque feuille de l'arbre (i.e., sur des affectations complètes), mais aussi sur divers nœuds de l'arbre de recherche (i.e., sur des affectations partielles)
- effectuer une recherche locale utilisant des techniques de recherche systématique pour choisir un voisin ou pour filtrer l'espace de recherche

La plupart des travaux récents tombent dans la troisième catégorie, qui, en combinant la puissance de la propagation des contraintes et les performances de la recherche locale, montrent de bons résultats sur les problèmes d'optimisation comme la planification ou l'ordonnancement. Cependant, tous ces algorithmes restent par nature incomplets et ne peuvent être utilisés pour résoudre des problèmes incohérents.

L'approche hybride que nous présentons dans le présent article tombe dans la première catégorie, avec quelques élements des autres sections. Ainsi, notre idée principale reste d'alterner entre une recherche locale et une recherche systématique, ce qui appartient clairement à la première catégorie. En apprenant et en conservant des informations sur le problème d'une exécution à l'autre, un algorithme peut guider l'autre, et vice versa. Cet apprentissage appartient plutôt aux deux autres catégories. Un des principaux avantages de la première catégorie est que l'on se base sur des algorithmes longuement étudiés, et que toute amélioration apportée à l'une des techniques va profiter à l'ensemble du système.

L'approche proposée dans [15] est particulièrement intéressante, puisque, comme la notre, elle tombe dans la première catégorie. Mazure et al. choisissent d'effectuer plusieurs exécutions d'une recherche locale en utilisant l'algorithme TSAT (un algorithme de recherche Tabu pour le problème SAT), avec un nombre limité de *flips* et *tries*. Si TSAT échoue, des statistiques obtenues sur les contraintes falsifiées pendant la recherche sont utiliseés pour résoudre le problème via un algorithme complet, ainsi que pour extraire un noyau inconsistant. Cependant, contrairement à notre algorithme, TSAT n'utilise pas directement le poids des contraintes, et l'heuristique utilisée par la recherche systématique n'est pas adaptatif. Tout en confirmant les résultats de Mazure et al. appliqués aux CSP, nous pensons donc que notre approche est à la fois plus naturelle et plus robuste.

Pour montrer l'efficacité de la pondération de contraintes pour identifier les parties plus difficiles des problèmes, nous avons essayé de lancer chaque algorithme sur des instances incohérentes du problème *Queens-Knights*. Il s'agit d'une fusion du problème cohérent des Reines avec le problèmes incohérent des Cavaliers, comme décrit dans [2]. Le problème constitué par 50 reines et 5 cavaliers est modélisé par 55 variables ($d = 2\ 500$) et 1 235 contraintes. Il contient un noyau minimalement inconsistent (MUC) de 5 variables et 5 contraintes. Après une première exécution de WMC avec 50 000 itérations, les 5 contraintes impliquées dans le MUC obtiennent un poids de 5 127 ou 5 128. Toutes les autres contraintes ont un poids inférieur ou égal à 2. Si l'on effectue une simple exécution de MGAC-$dom/wdeg$ avec 4 000 retour-arrières (ce qui consomme la même quantité de ressources que 50 000 itérations de WMC sur ce problème), les 5 contraintes du noyau ont respectivement un poids de 1 891, 1 765, 120, 1 et 1. Trois autres contraintes obtiennent un poids de 2 et les autres contraintes gardent leur poids initial de 1.

Cette expérimentation montre que WMC peut être bien plus efficace que MGAC-$dom/wdeg$ pour identifier des noyaux incohérents, et prouve l'efficacité potentielle de notre approche. Résoudre le problème des *Queens-Knights* avec notre prouveur hybride est fait en une exécution de WMC de 100 secondes suivi d'une exécution de MGAC-$dom/wdeg$ de 78s (2 468 retour-arrières). L'algorithme MGAC-$dom/wdeg$ seul effectue 6 exécutions pour un total de 328s et 12 652 retour-arrières.

5 Expérimentations

Les expérimentations sont effectuées par notre prouveur CSP4J, conçu en langage Java 5, sur un cluster de machines utilisant la JVM Sun 1.5.0.6 pour Linux, chacune équipée de deux processeurs x86-64 cadencés à 3 GHz et 2 Gio de RAM. Notre prouveur n'est pas conçu pour tirer partie de multiples processeurs, donc une seule instance a été résolue à la fois sur une machine. Nous avons comparé les performances des algorithmes suivants :

- MGAC-$dom/wdeg$ avec redémarrages seul (en augmentant à chaque redémarrage $maxBT$ de 50%) et apprentissage de nogoods comme décrit en section 3.3. Notre algorithme MGAC est basé sur l'algorithme d'établissement de l'arc consistance généralisée GAC3^{rm} [13]
- MCRW avec p fixé à 0,4 et $150\,000$ itérations par tour
- Tabu avec une liste Tabu fixée à 30 élements et $150\,000$ itérations par tour
- WMC avec $2\,000$ itérations par tour
- Notre approche hybride avec $2,000$ itérations par tour pour WMC, et avec $maxBT$ et $maxTries$ augmentés de 50% à chaque tour ($\alpha = 1,5$).

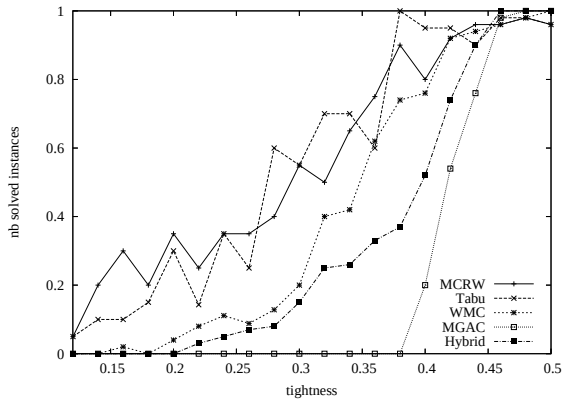

FIG. 3 – Résolution d'instances aléatoires

Les paramètres choisis sont optimaux pour des problèmes aléatoires binaires de 50 variables de 23 valeurs. Les problèmes aléatoires, de nature très homogène, sont une bonne référence pour estimer le comportement asymptotique des algorithmes. Nous avons comparé MGAC-$dom/wdeg$, WMC ainsi que notre algorithme hybride sur une série de problèmes aléatoires difficiles présentant des duretés variées. Les problèmes aléatoires binaires sont générés en fonction d'un quadruplet $\langle n, d, t, e\rangle$, respectivement le nombre de variables, la taille des domaines, la dureté des contraintes et le nombre de contraintes. La dureté des contraintes représente le nombre de couples de valeurs autorisés par rapport au nombre de couples possibles (d^2). Nous fixons $n = 50$ et $d = 23$ et faisons varier t et e de telle sorte que tous les problèmes soient situés au seuil ($e = -\frac{n \ln d}{\ln(1-t)}$). Quand la dureté des contraintes diminue, le nombre de variables (et donc la densité du graphe) augmente, et vice versa. Pour que le théorème permettant le calcul du seuil, la dureté des contraintes doit être inférieure à $0,5$. On peut également forcer les problèmes à être cohérents sans modifier la difficulté inhérente de ceux-ci [24]. *Toutes les instances sont au seuil et cohérentes.*

Les résultats obtenus dont montrés sur la Figure 3. Ils montrent que, bien que MGAC-$dom/wdeg$ soit extrêmement puissant sur les problèmes peu denses, aux contraintes dures, les algorithmes de recherche locale sont plus efficace à l'opposé, sur les problèmes denses aux contraintes peu dures. Ce sont aussi des problèmes considérés comme plus difficiles. Sur des problèmes aussi homogènes que ces instances aléatoires, la pondération de contraintes est pratiquement sans effet. Les performances décevantes de l'algorithme hybride sont donc compréhensibles. Cependant, on peut constater que l'algorithme hybride est plus robuste, dans le sens où le nombre global d'instances résolues est plus important que pour MGAC-$dom/wdeg$ seul, tout en restant susceptible de résoudre des instances incohérentes. D'autre part, la mesure de la densité ou de la dureté n'est pas fiable pour des problèmes structurés hétérogènes, puisqu'ils peuvent être composés de plusieurs sous-problèmes difficiles ayant chacun une valeur différente de dureté ou de densité. On ne peut donc utiliser ces mesures, dans l'état actuel des choses, pour sélectionner le meilleur algorithme pour un problème donné. L'utilisation d'algorithmes hybrides robustes est donc particulièrement intéressante dans un environnement "boîte noire", où l'on ne peut compter sur aucune information sur la nature des problèmes.

Nous avons également exécuté les différents algorithmes sur toutes les instances de la Seconde Compétition Internationale de Prouveurs CSP [1]. Tous les benchmarks sont disponibles en ligne dans le format XCSP 2.0 sur le site de la compétition. Les algorithmes de recherche locale n'étant pas conçus pour résoudre les problèmes impliquant des contraintes d'arité élevée (ce qui nécessiterait un traitement non booléen des contraintes [7]), les problèmes contenant des contraintes d'arité supérieure à 4 ont été ignorés. Nous avons conservé les instances pour lesquelles au moins un des algorithmes a pu trouver une solution, et avons éliminé les instances incohérentes. Les résultats obtenus sont reportés dans la Table 1. Bien que l'algorithme MGAC-$dom/wdeg$ se révèle très robuste face aux algorithmes de recherche locale, les problèmes d'atelier (job-shop et open-shop) ainsi que les problèmes aléatoires difficiles comportant beaucoup de variables ($n > d$) sont plus efficacement résolus par la recherche locale. WMC se montre particulièrement efficace face à MCRW et la recherche Tabu quand les paramètres de ceux ci sont inadaptés au problème considéré (aim, fapp, ruler...). L'algorithme hybride parvenient à combiner la puissance de MGAC-$dom/wdeg$ et de WMC en réussissant à résoudre globalement le plus grand nombre de problèmes structurés.

La Table 2 montre le comportement de MGAC-$dom/wdeg$ et de notre prouveur hybride sur une série de problèmes industriels (RLFAP, Open-Shop) et académiques incohérents. Dans tous les cas, le nombre global d'affectations pour MGAC-$dom/wdeg$ est beaucoup plus faible en coopération avec WMC que seul, ce qui prouve l'efficacité des techniques d'apprentissage proposées. Dans la plupart des cas, le temps CPU global est également meilleur. Parfois, en particulier sur les instances faciles ou homogènes, trop de temps est perdu lors de la recherche locale (sur *scen11-f8*, *os-95-5* ou *os-95-8* par exemple) par rapport au temps gagné sur MGAC-$dom/wdeg$. Nous pensons pouvoir améliorer ces cas en utilisant des meilleurs paramètres pour la recherche locale (en particulier le nombre maximal d'itérations) ou en effectuant des optimisations dans l'implantation de WMC.

6 Conclusion et perspectives

Dans cet article, nous avons introduit une approche hybride entre la recherche systématique et la recherche locale,

		MGAC-$dom/wdeg$		MCRW		Tabu		WMC		Hybrid	
$instance$	nb	$solved$	$time$	$solved$	$time$	$solved$	$time$	$solved$	$time$	$solved$	$time$
all interval	13	12	**0.44**	**13**	0.89	12	0.57	**13**	0.68	**13**	2.45
qcp-qwh-bqwh	253	248	0.78	**253**	0.60	**253**	**0.41**	246	0.76	**253**	2.11
aim	96	**94**	0.52	64	8.95	63	17.2	86	**0.50**	82	0.85
fapp	147	141	11.20	**142**	4.20	62	>600.00	132	**2.44**	141	125.77
ruler	9	**8**	17.42	5	59.34	5	**8.87**	6	12.00	**8**	9.98
shop	127	103	1.91	76	46.08	56	>600.00	**110**	**1.18**	104	6.33
par	18	**16**	0.89	10	**0.56**	10	**0.56**	10	**0.56**	14	7.24
queens	20	17	5.92	18	0.98	**19**	0.92	18	**0.84**	**19**	32.54
ramsey	11	9	4.68	**11**	2.60	9	3.54	**11**	**1.71**	**11**	22.28
rlfap	25	**25**	2.68	**25**	1.97	19	1.71	21	**1.14**	24	0.96
rand-d>n	94	**94**	3.89	31	>600.00	16	>600.00	33	>600.00	42	>600
rand-d<n	672	561	22.32	550	37.98	**592**	**19.7**	589	22.62	582	22.41

TAB. 1 – Resultats sur des instances satisfiables : nombre d'instances résolues et temps médian en secondes

	pure MGAC-$dom/wdeg$			hybrid					
instance	$runs$	$assgns$	CPU	WMC		MGAC-$dom/wdeg$			$total\ CPU$
				$runs$	CPU	$runs$	$assgns$	CPU	
scen11-f8	1	13 596	**107,9**	1	126,4	1	**470**	19,1	145,5
scen11-f7	2	38 692	268,4	1	124,8	1	**2 924**	31,9	**156,7**
scen11-f6	2	40 323	293,5	1	129	1	**6 894**	62,1	**191,1**
scen11-f5	3	68 307	713,3	2	258,3	2	**47 692**	378,6	**634,3**
os-5-95-2	7	79 599	105,8	1	20,8	1	**3 148**	10,7	**31,5**
os-5-95-5	4	30 262	**52,1**	2	39,5	2	**11 372**	29,2	68,7
os-5-95-8	4	24 137	**55,5**	2	47	2	**12 543**	30,2	77,2
qK-50-5-add	6	12 652	328,0	2	188	2	**2 495**	79,3	**267,3**
qK-50-5-mul	7	13 482	**452,3**	2	398	2	**2 495**	73,0	525,3
qK-80-5-add	> 10	$> 38\,000$	$> 6\,000,0$	3	854,9	3	**6 395**	603,1	**1 558,0**
qK-80-5-mul	> 9	$> 25\,000$	$> 6\,000,0$	2	1 666,2	2	**6 395**	971,6	**2 637,8**

TAB. 2 – Résultats sur plusieurs problèmes difficiles

respectivement représentées par les algorithmes MGAC-$dom/wdeg$ et WMC. Nous proposons un système d'apprentissage pour améliorer les performances des deux algorithmes. Des expérimentations avec nos implantations de MGAC-$dom/wdeg$, MCRW, recherche Tabu et WMC, basées sur les dernières mises aux point publiées, et sur de nombreux benchmarks disponibles ont montré la robustesse de notre approche, avec de bons résultats en particulier sur les instances structurées, bien que leur taille et leur nature soient très variées.

Tout en confirmant les résultats obtenus par Mazure et al. sur le problème SAT [15], nous améliorons les relations entre les deux algorithmes en liant de manière très naturelle les heuristiques utilisées par chaque algorithme. Ceci nous a mené à une stratégie de recherche robuste où l'impact de la recherche locale et systématique peut être finement paramétrée.

D'autres informations pourraient également être extraites d'une exécution d'un algorithme, et un paramétrage adaptatif des algorithmes pourrait améliorer encore la robustesse de la recherche. La possibilité de lancer plusieurs algorithmes en parallèle doit également être prise en considération.

Remerciements Je voudrais remercier Christophe Lecoutre et Lakhdar Saïs pour leurs commentaires constructifs et pour leurs suggestions qui ont permis d'améliorer grandement la qualité globale du présent article.

Références

[1] Second International CSP Solvers Competition. http ://www.cril.univ-artois.fr/CPAI06, 2006.

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

[3] H. Cambazard and N. Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. *Constraints*, 2006.

[4] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[5] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming, 2001.

[6] P. Galinier and J.K. Hao. Tabu search for maximal constraint satisfaction problems. *Proceedings of CP'97*, pages 196–208, 1997.

[7] P. Galinier and J.K. Hao. A General Approach for Constraint Solving by Local Search. *Journal of Mathematical Modelling and Algorithms*, 3(1) :73–88, 2004.

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

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

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

[11] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. In *Seventh National Conference on Artificial Intelligence – AAAI'2000*, pages 169–174, Austin, TX, USA, August 2000.

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

[13] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'2007)*, pages 125–130, 2007.

[14] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'2007)*, 2007.

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

[16] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts : a heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3) :161–205, 1992.

[17] P. Morris. The breakout method for escaping from local minima. In *Proceedings of AAAI'93*, pages 40–45, 1993.

[18] S. Prestwich. Local search and backtracking versus non-systematic backtracking. In *Papers from the AAAI 2001 Fall Symposium on Using Uncertainty Within Computation*, pages 109–115, North Falmouth, Cape Cod, MA, November 2-4 2001. The American Association for Artificial Intelligence, The AAAI Press.

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

[20] A. Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In *Proceedings of IJCAI'97*, pages 1254–1259, 1997.

[21] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. *Proc. IJCAI'97*, 1997.

[22] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

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

[24] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. A simple model to generate hard satisfiable instances. In *Proceedings of IJCAI'05*, pages 337–342, 2005.