Résolution Etendue par Substitution Dynamique des Fonctions Booléennes

Said Jabbour, Jerry Lonlac, Lakhdar Sais

Résolution Etendue par Substitution Dynamique des Fonctions Booléennes

Said Jabbour[1] **Jerry Lonlac**[1,2] **Lakhdar Saïs**[1]

[1] CRIL - CNRS - Université Lille-Nord de France F-62307 Lens Cedex
[2] Département d'Informatique - Université de Yaoundé 1 B.P. 812 Yaoundé, Cameroun
{jabbour,lonlac,sais}@cril.fr

Abstract

Ce travail présente une technique originale de substitution dynamique des fonctions booléennes. Il détecte premièrement un ensemble de fonctions booléennes dans la formule booléenne sous forme normale conjonctive (CNF). Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises en substituant les arguments d'entrée par les arguments de sortie. Ceci conduit à une façon originale d'intégrer une forme restreinte de résolution étendue, l'un des plus puissants systèmes de preuve par résolution. En effet, la plupart des fonctions booléennes extraites correspondent à celles introduites au cours de la phase d'encodage CNF en appliquant le principe d'extension Tseitin. Substituer les entrées par les sorties peut être vu comme une façon élégante d'imiter la résolution étendue avec la manipulation des sous-formules. Des expérimentations préliminaires montrent la faisabilité de notre approche sur certaines classes d'instances SAT prises des récentes compétitions SAT et SAT-Race.

1 Introduction

Le problème SAT, i.e., la vérification de la fatisfaction d'un ensemble de clauses, est central dans plusieurs domaines et notamment en intelligence artificielle incluant le problème de satisfaction de contraintes (CSP), planification, raisonnement non-monotone, vérification de logiciels, etc. Aujourd'hui, SAT a gagné une audience considérable avec l'apparition d'une nouvelle génération de solveurs SAT capable de résoudre de très grandes instances issues du codage des applications du monde réel ainsi que par le fait que ces solveurs constituent d'importants composants de base pour plusieurs domaines, e.g., SMT (SAT modulo théorie), preuve de théorème, comptage de modèles, problème QBF, etc. Ces solveurs appelés solveurs SAT modernes [23, 13], sont basés sur la propagation de contraintes classiques [11] éfficacement combinée à d'éfficaces structures de données avec : (i) stratégies de redémarrage [16, 19], (ii) activité basée sur l'heuristique de choix de variables (comme VSIDS) [23], et (iii) apprentissage de clauses [22, 6, 23]. l'apprentissage de clauses est maintenant connu comme l'un des éléments les plus importants des solveurs SAT modernes. L'idée principale est que quand une branche courante de l'arbre de recherche conduit à un conflit, l'apprentissage de clauses vise à dériver une clause qui succinctement exprime les causes du conflit. Cette clause apprise est ensuite utilisée pour élaguer l'espace de recherche. L'apprentissage de clauses également connu dans la littérature comme "Conflict Driven Clause Learning" (CDCL) se réfère maintenant au plus connu et utilisé schéma d'apprentissage premier UIP, d'abord intégré dans les solveurs SAT Grasp [22] et éfficacement mise en œuvre dans zChaff [23]. Les solveurs SAT modernes peuvent être vus comme une extension de la bien connue procédure DPLL classique obtenue grâce à différentes améliorations. Il est important de noter que la bien connue règle de résolution joue encore un rôle prépondérant dans l'efficacité des solveurs SAT modernes qui peut être vu comme une forme particulière de la résolution générale [7].

Théoriquement, en intégrant l'apprentissage de clause à la procédure DPLL classique [11], le solveur SAT obtenu formulé comme un système de preuve est aussi puissant que la résolution générale [25]. Ce résultat théorique important, suggère que, pour améliorer l'efficacité des solveurs SAT, on a besoin de trouver des systèmes de preuves plus puissants. Cette question de recherche est également motivée par l'existence de plus en plus des applications challengeant les ins-

tances SAT qui ne peuvent actuellement être résolues par les solveurs SAT disponibles. Le résolution étendue [27] et la résolution symétrie [21] sont les deux systèmes de preuve qui sont connus comme étant plus puissant que la résolution. Le principe d'extension permet l'introduction des variables auxiliaires pour représenter les formules intermédiaires de telle sorte que la longueur d'une preuve peut être considérablement réduite en manipulant ces variables au lieu de manipuler les formules qu'elles représentent. La symétrie, d'autre part, permet de reconnaître qu'une formule reste invariante sous certaines permutations de noms de variables, et utilise cette information pour éviter de répéter les dérivations indépendantes des formules intermédiaires qui ne sont que des variantes permutationnelles d'une autre formule.

Pour la symétrie, plusieurs approches ont été proposées à la fois dans la satisfiabilité booléenne (eg. [10, 8, 1]) et dans la programmation par contraintes (e.g. [26, 15, 14]). Cependant, l'automatisation du système de preuve par résolution étendue reste encore une question ouverte. La résolution étendue ajoute une règle d'extension au système de preuve par résolution, permettant l'introduction de définitions dans la preuve. Par exemple étant donnée une formule booléenne $\mathcal{F}$, contenant les variables a et b, et v une nouvelle variable, on peut ajouter la définiton $v \leftrightarrow a \vee b$ tout en préservant la satisfiabilité. Rappelons que la règle d'extension est à la base de la transformation linéaire des formules booléennes générales en forme normale conjonctive (CNF). Par exemple, pour $n > 4$, la formule $\mathcal{F} = (x_1 \leftrightarrow x_2 \leftrightarrow \cdots \leftrightarrow x_n)$ n'a aucune formule polynomialement équivalente dans la représentation $CNF(\mathcal{F})$. Cependant, en utilisant le principe d'extension, on peut obtenir une transformation linéaire de $\mathcal{F}$ à $CNF(\mathcal{F})$. La formule CNF obtenue est équivalente à celle d'origine par rapport à la satisfiabilité. Il est également démontré que certaines formules qui sont difficiles pour la résolution (e.g. pigeon-hole formulas [18]) admettent des courtes preuves par résolution étendue [9]. La courte preuve pour le problème des pigeons est obtenue par simulation à l'aide de l'induction de la règle d'extension. Comme mentionné par B. Krishnamurthy dans [20], l'extension permet la définition d'hypothèses intermédiaires qui peuvent être introduites par la définition de nouvelles variables reposant sur ces hypothèses. En manipulant ces hypothèses, de courtes preuves peuvent être obtenues pour certaines formules difficiles.

Récemment, un solveur SAT CDCL basé sur une restriction de l'application de la règle d'extension a été proposé dans [3]. Plus précisement, quand 2 clauses successives apprises par le solveur sont de la forme $l_1 \vee \alpha$, $l_2 \vee \alpha$, une nouvelle variable $z \leftrightarrow \neg l_1 \vee \neg l_2$) est introduite.

La principale difficulté derrière l'automatisation de la résolution étendue est de déterminer (1) quand est ce que de telles définitions seront ajoutées et (2) quelles fonctions booléennes représenteront elles. L'approche proposée dans ce papier exploite les nouvelles variables introduites dans la phase d'encodage des formules booléennes générales pour CNF grâce à l'application du principe d'extension de Tseitin. Substituer ces variables pendant la recherche peut être vu comme une façon élégante de manipuler dynamiquement ces sous-formules. Cela imite clairement le principe de la résolution étendue. Notre proposition répond à la fois aux questions (1) et (2), en exploitant les fonctions booléennes cachées généralement introduites lors de l'encodage, et en les utilisant afin de minimiser les clauses apprises. De plus, comme on peut le voir plus tard, notre approche proposée à une certaine similitude et différence avec le raisonnement binaire et la règle d'hyper résolution [5, 4].

Le papier est organisé comme suit. Après quelques notations et définitions préliminaires, quelques notions théoriques autours des solveurs SAT, fonctions booléennes (section 2), notre substitution dynamique des fonctions booléennes est décrite dans la section 3. Finalement, avant de conclure, les résultats expérimentaux démontrant la faisabilité de notre approche sont présentés.

2 Définitions

2.1 Définitions et notations préliminaires

Une *formule CNF* $\mathcal{F}$ est une conjonction de *clauses*, où une clause est une disjonction de *littéraux*. Un littéral est interprété comme une variable propositionnelle positive (x) ou negative ($\neg x$). Les deux littéraux x et $\neg x$ sont appélés *complémentaire*. On note par $\bar{l}$ le littéral complémentaire de l. Pour un ensemble de littéraux L, $\bar{L}$ est défini comme $\{\bar{l} \mid l \in L\}$. Une clause *unitaire* est une clause contenant seulement un seul littéral (appelé *littéral unitaire*), tandis qu'une clause binaire contient excatement deux littéraux. Une *clause vide* , notée $\bot$, est interprétée comme fausse (insatisfiable), alors qu'une *formule CNF vide*, notée $\top$, est interprétée comme vraie (satisfiable).

L'ensemble des variables apparaissant dans $\mathcal{F}$ est noté $V_{\mathcal{F}}$. Un ensemble de littéraux est *complet* s'il contient un littéral pour chaque variable de $V_{\mathcal{F}}$, et *fondamental* s'il ne contient pas de littéraux complémentaires. Une *affectation* ρ d'une formule booléenne $\mathcal{F}$ est une fonction qui associe la valeur $\rho(x) \in \{false, true\}$ à quelques variables de $x \in \mathcal{F}$. ρ est *complète* s'il attribue une valeur pour chaque

variable $x \in \mathcal{F}$, et *partielle* sinon. Une affectation est alternativement représentée par un ensemble de littéraux complet et fondamental, de façon évidente un *modèle* d'une formule $\mathcal{F}$ est une affectation ρ qui laisse la formule *vraie* ; notée $\rho \models \Sigma$.

On dénote par $\eta[x, c_i, c_j]$ la *résolvante* entre une clause c_i contenant le littéral x et c_j une clause contenant l'opposé de ce même littéral $\neg x$. En d'autres termes, $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$.

2.2 Recherche DPLL

DPLL [11] est une procédure de recherche de type *backtrack*; À chaque nœud les littéraux affectés (le littéral de décision et les littéraux propagés) sont étiquetés avec le même *niveau de décision*, initialisé à 1 et incrémenté à chaque nouveau point de décision. Le niveau de décision courant est le niveau le plus élevé dans la pile de propagation. Lors d'un retour-arrière ("*backtrack*"), les variables ayant un niveau supérieur au niveau du backtrack sont défaites et le niveau de décision courant est décrémenté en conséquence (égal au niveau du backtrack). Au niveau i, l'interprétation partielle courante ρ peut être représentée comme une séquence de décision-propagations de la forme $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ où le premier littéral x_k^i correspond au littéral de décision x_k affecté au niveau i et chaque $x_{k_j}^i$ de l'ensemble $1 \leq j \leq n_k$ représente les littéraux unitaires propagés à ce même niveau i. Soit $x \in \rho$, On note $l(x)$ le niveau d'affectation de x. Pour une clause α, $l(\alpha)$ est défini comme le maximum des niveaux de ses littéraux affectés.

2.3 Fonctions Booléennes

Une fonction Booléenne est une expression de la forme $y = f(x_1, \ldots, x_n)$, où $f \in \{\vee, \wedge\}$ et où y et x_i sont des littéraux, qui est définie comme suit :

- $y = \wedge(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{y \vee \neg x_1 \vee \cdots \vee \neg x_n, \neg y \vee x_1, \ldots, \neg y \vee x_n\}$, traduisant la condition selon laquelle la valeur de vérité de y est determinée par la conjonction des valeurs de vérité des x_i t.q.$i \in [1 \ldots n]$;
- $y = \vee(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{\neg y \vee x_1 \vee \cdots \vee x_n, y \vee \neg x_1, \ldots, y \vee \neg x_n\}$;

Dans la suite, nous considérons seulement les fonctions booléennes de la forme $y = \vee(x_1, \ldots, x_n)$ où y et x_i sont des variables booléennes de la formule originale. En effet, une fonction booléenne $y = \wedge(x_1, \ldots, x_n)$ peut être écrite comme $\neg y = \vee(\neg x_1, \ldots, \neg x_n)$.

Pour une fonction boolénne de la forme $y = f(x_1^{'}, \ldots, x_n^{'})$, où $x_i^{'} \in \{x_i, \neg x_i\}$, la variable y est une variable de sortie tandis que les variables $x_1, \ldots, x_n$ sont appelées variables d'entrée. Ces fonctions nous permettent de détecter un sous-ensemble de variables dépendantes (i.e. variables de sortie) dont la valeur de vérité dépend des valeurs de vérité des variables d'entrée.

2.4 Detection des fonctions booléennes dans une formule CNF

Etant donné une formule CNF $\mathcal{F}$, On peut utiliser deux méthodes différentes pour la détection des fonctions booléennes encodées par les clauses de $\mathcal{F}$.
La première méthode de détection, appelée méthode syntaxique, est une approche de type *pattern matching* qui nous permet de detecter les fonctions booléennes qui apparaissent directement dans la structure de la formule CNF [24].

Exemple 1 *Soit* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg y \vee x_2), (\neg y \vee x_3)\}$.

Dans cet exemple, nous pouvons détecter syntaxiquement la fonction $y = \wedge(x_1, x_2, x_3)$.

La seconde méthode est l'approche de détection sémantique où les fonctions sont détectées en utilisant la propagation unitaire (UP) [17]. En effet, cette méthode nous permet de détecter les fonctions booléennes cachées.

Exemple 2 *Let* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg x_1 \vee x_4), (\neg x_4 \vee x_2), (\neg x_2 \vee x_5), (\neg x_4 \vee \neg x_5 \vee x_3)\}$.

Dans cet exemple, $UP(\mathcal{F} \wedge y) = \{x_1, x_4, x_2, x_5, x_3\}$ l'ensemble des littéraux obtenus par propagation unitaire (UP) et nous avons la clause $c = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3) \in \mathcal{F}$ qui est telle que $c \backslash \{y\} \subset \overline{UP(\mathcal{F} \wedge y)}$. Ainsi, nous pouvons detecter la fonction booléenne $y = \wedge(x_1, x_2, x_3)$, que nous ne pouvons pas détecter en utilisant la méthode syntaxique ci-dessus.

Dans la suite, étant donnée une formule CNF $\mathcal{F}$, nous utilisons ces deux méthodes pour détecter toutes les fonctions booléennes encodés par les clauses de $\mathcal{F}$. De toute évidence, l'ensemble des fonctions booléennes qui peuvent être détectées par l'approche sémantique est une extension de l'ensemble des fonctions booléennes extraites à l'aide de la l'approche syntaxique.

Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises durant la recherche. Dans notre implémentation, nous exploitons les deux approches de détections des fonctions booléennes proposées dans [24] et [17].

2.5 Analyse de conflit et graphe d'implications

Le graphe d'implications est une représentation standard utilisée classiquement pour analyser les conflits dans les solveurs **SAT** modernes. À chaque fois qu'un littéral y est propagé, on sauvegarde une référence de la clause impliquant la propagation de y, qu'on note $imp(y)$. La clause $imp(y)$, appelée implication de y, est dans ce cas de la forme $(x_1 \vee \cdots \vee x_n \vee y)$ où chaque littéral x_i est faux sous l'affectation partielle courante $(\rho(x_i) = false, \forall i \in 1..n)$, alors que $\rho(y) = vraie$. lorsqu'un littéral y n'est pas obtenu par propagation mais issue d'une décision, $imp(y)$ est indéfinie, qu'on note par convention $imp(y) = \perp$. Quand $imp(y) \neq \perp$, on note par $exp(y)$ l'ensemble $\{\overline{x} \mid x \in imp(y) \setminus \{y\}\}$, appelé ensemble des *explications* de y. Quand $imp(y)$ est indéfini, on défini $exp(y)$ comme l'ensemble vide.

Définition 1 (Graphe d'implications) *Soit $\mathcal{F}$ une formule CNF, ρ une affectation partielle, et soit exp l'ensemble des explications pour les littéraux déduits (propagés unitairement) dans ρ. Le graphe d'implications associé à $\mathcal{F}$, ρ et exp est $\mathcal{G}_{\mathcal{F}}^{\rho,exp} = (\mathcal{N}, \mathcal{E})$ où :*

- *$\mathcal{N} = \rho$, i.e., il existe un nœud pour chaque littéral de décision ou propagé ;*
- *$\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in \exp(y)\}$*

Dans le reste du papier, pour des raisons de simplicité, un graphe d'implication est tout simplement noté $\mathcal{G}_{\mathcal{F}}^{\rho}$. On note aussi m le niveau du conflit.

Exemple 3 *$\mathcal{G}_{\mathcal{F}}^{\rho}$, montré dans Figure 1 représente le graphe d'implications de la formule $\mathcal{F}$ et l'affectation partielle ρ donnée ci-dessous : $\mathcal{F} \supseteq \{c_1, \ldots, c_{12}\}$*

$(c_1) \neg x_1 \vee \neg x_{11} \vee x_2$	$(c_2) \neg x_1 \vee x_3$
$(c_3) \neg x_2 \vee \neg x_{12} \vee x_4$	$(c_4) \neg x_1 \vee \vee \neg x_3 \vee x_5$
$(c_5) \neg x_4 \vee \neg x_5 \vee \neg x_6 \vee x_7$	$(c_6) \neg x_5 \vee \neg x_6 \vee x_8$
$(c_7) \neg x_7 \vee x_9$	$(c_8) \neg x_5 \vee \neg x_8 \vee \neg x_9$
$(c_9) \neg x_{10} \vee \neg x_{17} \vee x_1$	$(c_{10}) \neg x_{13} \vee \neg x_{14} \vee x_{10}$
$(c_{11}) \neg x_{13} \vee x_{17}$	$(c_{12}) \neg x_{15} \vee \neg x_{16} \vee x_{13}$

$\rho = \{\langle \ldots x_{15}^1 \ldots \rangle \langle (x_{11}^2) \ldots \ldots \rangle \langle (x_{12}^3) \ldots x_6^3 \ldots \rangle \langle (x_{14}^4), \ldots \rangle \langle (x_{16}^5), x_{13}^5, \ldots \rangle\}$. Le niveau du conflit est 5 et $\rho(\mathcal{F}) = false$.

Définition 2 (Clause assertive) *Une clause c de la forme $(\alpha \vee x)$ est appelée clause assertive ssi $\rho(c) = false$, $l(x) = m$ et $\forall y \in \alpha, l(y) < l(x)$. x est appelé littéral assertif.*

L'analyse de conflits est le résultat de l'application de la résolution à partir de la clause conflit en utilisant différentes implications encodées dans le graphe d'implications. On appelle celà dérivation de la clause assertive (DCA en court).

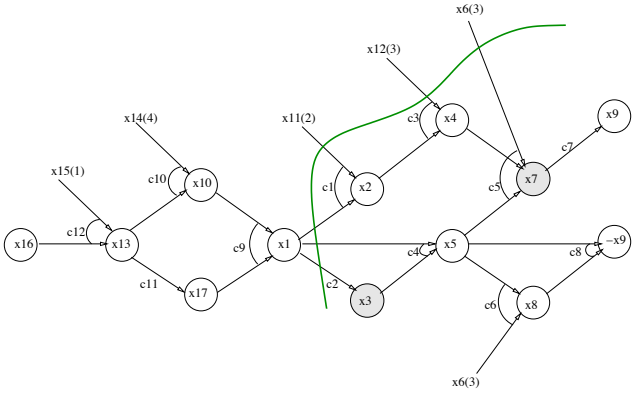

FIGURE 1 – Graphe d'implications $\mathcal{G}_{\mathcal{F}}^{\rho} = (\mathcal{N}, \mathcal{E})$

Définition 3 (dérivation de la clause assertive) *La dérivation de la clause assertive pi est une séquence de clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfaisant les conditions suivantes :*

1. *$\sigma_1 = \eta[x, imp(x), imp(\neg x)]$, tel que $\{x, \neg x\}$ est le conflit.*
2. *σ_i, pour $i \in 2..k$, est construite en selectionnant un littéral $y \in \sigma_{i-1}$ pour lequel $imp(\overline{y})$ est défini. Nous avons alors $y \in \sigma_{i-1}$ et $\overline{y} \in imp(\overline{y})$: les deux clauses résolues. La clause σ_i est définie comme $\eta[y, \sigma_{i-1}, imp(\overline{y})]$;*
3. *σ_k est en plus une clause assertive.*

considérons à nouveau l'exemple 3. Le parcours du graphe $\mathcal{G}_{\mathcal{F}}^{\rho}$ (voir Fig. 1) conduit à la dérivation de la clause assertive : $\langle \sigma_1, \ldots, \sigma_7 \rangle$ où $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_5}^5 \vee \neg {x_7}^5 \vee \neg {x_8}^5)$ et $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$. La clause σ_7 est la prémière clause rencontrée qui contient un seul littéral du niveau de décision courant. Notons que cette résolvante est fausse sous l'affectation ρ. Par conséquent, le littéral $\neg x_1$ est impliqué au niveau 3. Les solveurs SAT ajoutent la clause (σ_7) à la base des clauses apprises, retourne au niveau 3 et affecte le littéral assertif $\neg x_1$ à la valeur de vérité vraie. Le nœud x_1 correspondant au littéral assertif $\neg x_1$ est appelé *First Unique Implication Point (First UIP)*. Pour plus d'informations autours des solveurs CDCL basés sur les clauses apprises, nous reférons les lecteurs à [22, 23, 2].

3 Substitution dynamique des fonctions booléennes

Dans cette section, nous montrons comment les fonctions booléennes détectées dans la formule origi-

nale peuvent être utilisées pour réduire dynamiquement la taille des différentes clauses apprises.

3.1 Exemple

Nous illustrons cette nouvelle approche de substitution dynamique des fonctions booléennes en utilisant un simple exemple.

Exemple 4 *Soit $\mathcal{F}$ la formule CNF précédente, supposons que $\mathcal{F}$ contient aussi les clauses suivantes $\{c_{13}, \ldots, c_{16}\}$: $\mathcal{F} \supseteq \{c_{13}, \ldots, c_{16}\}$*

$(c_{13}) \neg y \vee \neg x_{11} \vee \neg x_{12} \vee \neg x_6$
$(c_{14}) \quad y \vee x_{11}$
$(c_{15}) \quad y \vee x_{12}$
$(c_{16}) \quad y \vee x_6$

Soit $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$ la clause apprise précédente après l'analyse de conflit. Les clauses c_{13}, c_{14}, c_{15} et c_{16} encodent la foncion booléenne $y = \vee(\neg x_{11}, \neg x_{12}, \neg x_6)$. Comme on peut le voir, la clause apprise σ_7 contient les entrées de la fonction boolénne précédente. Par conséquent, on peut remplacer ces entrées par la sortie y dans σ_7. Cette substitution permet d'éliminer les littéraux $\neg x_{11}, \neg x_{12}, \neg x_6$ de la clause apprise σ_7.
Après ce processus, on obtient $\sigma'_7 = (\neg x_1 \vee y)$. Il est important de noter que la résolvante σ'_7 peut être obtenue par hyper résolution binaire entre les clauses binaires c_{14}, c_{15} et c_{16} et la clause c_{13}. La principale différence est que dans notre approche, y doit être la sortie d'une fonction booléenne.

Exemple 5 *Soit $\mathcal{F}$ la formule CNF précédente, supposons maintenant que $\mathcal{F}$ contient plutôt les clauses suivantes $\{c_{13}, \ldots, c_{17}\}$: $\mathcal{F} \supseteq \{c_{13}, \ldots, c_{17}\}$*

$(c_{13}) \quad y \vee \neg x_{12} \vee \neg x_6$
$(c_{14}) \quad \neg y \vee x_{14}$
$(c_{15}) \quad \neg y \vee x_{15}$
$(c_{16}) \quad \neg x_{14} \vee \neg x_{15} \vee x_{12}$
$(c_{17}) \quad \neg x_{12} \vee \neg x_{14} \vee x_6$

Dans ce deuxième exemple, on détecte par propagation unitaire la fonction booléenne $y = \wedge(x_{12}, x_6)$ qui est équivalente à $\neg y = \vee(\neg x_{12}, \neg x_6)$. En effet, $UP(\mathcal{F} \wedge y) = \{x_{14}, x_{15}, x_{12}, x_6\}$. Les littéraux $\neg x_6$ et $\neg x_{12}$ sont éliminés de la clause apprise σ_7 et remplaçés par la sortie $\neg y$. Après cette substitution, on obtient $\sigma''_7 = (\neg y \vee \neg x_{11} \vee \neg x_1)$.

La méthode de détection basée sur la propagation unitaire nous permet de détecter toutes les fonctions booléennes que nous ne pouvons pas détecter syntaxiquement. Il est important de noter que lorsque les fonctions booléennes sont détectées en utilisant la propagation unitaire, la résolvante σ''_7 ne peut pas être obtenue directement par hyper-résolution binaire. Pour que cela soit possible, on a besoin premièrement de générer les clauses $(\neg y \vee x_{12})$ et $(\neg y \vee x_6)$ et par résolution avec les clauses c_{14}, c_{15}, c_{16} et c_{17}.

Notons que plusieurs entrées peuvent être substituées par la même sortie. Ceci arrive lorsque nous avons plusieurs fonctions booléennes dans la formule originale avec la même sortie. Dans ce cas, toutes les différentes entrées incluses dans la clause apprise sont éliminées mais cette sortie n'est ajoutée qu'une seule fois dans la clause apprise.

Les fonctions booléennes utilisées ici pour réduire les clauses apprises sont encodées par les clauses de la formule originale. Ainsi, aucune variable additionnelle n'est ajoutée à la formule comme dans le cas des systèmes de preuves par résolution étendue. Cependant, certaines fonctions booléennes encodées dans la formule CNF sont introduites par le principe d'extension utilisé dans la phase d'encodage CNF.

3.2 Substitution dynamique

On donne à présent une présentation formelle de notre approche basée sur la substitution dynamique.

Proposition 1 *[Substitution dynamique] Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, où x est le littéral assertif et α une disjonction de littéraux. Soit $y = \vee(x_1, \ldots, x_n)$ avec $n \geq 2$, une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q. $\{x_1, \ldots, x_n\} \subseteq \alpha$, alors σ peut être réduite à σ' en substituant $\{x_1, \ldots, x_n\}$ par y dans α t.q. :*
- *$\sigma' = \beta \vee x$ si $y \in \alpha$*
- *$\sigma' = \beta \vee x \vee y$ sinon*
  avec $\beta = \alpha \backslash \{x_1, \ldots, x_n\}$

Propriété 1 *Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, où x est le littéral assertif. Soit $y = \vee(x_1, \ldots, x_n)$ une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q.$\{x_1, \ldots, x_n\} \subseteq \alpha$. on a :*
- *$\forall 1 \leq i \leq n, \rho(x_i) = faux$ et $\rho(y) = faux$*
- *$l(y) = max\{l(x_i), 1 \leq i \leq n\}$ et $l(y) < l(x)$*

Preuve 1 *Par définition de la clause apprise σ, on a $\rho(\sigma) = faux$. Comme $\{x_1, \ldots, x_n\} \subseteq \sigma$. Par conséquent, $\forall 1 \leq i \leq n, \rho(x_i) = faux$.*
On a $y = \vee(x_1, \ldots, x_n)$ alors $\rho(y) = faux$.
Par définition de la fonction booléenne $y = \vee(x_1, \ldots, x_n)$, $l(y)$ est le niveau du dernier littéral x_i qui fut affecté à faux. Par conséquent, $l(y) =$

$max\{l(x_i), 1 \leq i \leq n\}$. *Comme* x *est le littéral assertif, par définition,* $\forall z \in \alpha, l(z) < l(x)$. *Puisque* $\{x_1, \ldots, x_n\} \subseteq \alpha$ *et* $l(y) = max\{l(x_i), 1 \leq i \leq n\}$, *alors* $l(y) < l(x)$.

Propriété 2 *Soit* $\sigma = (\alpha \vee x)$ *une clause apprise, et* $G = \{g_1, \ldots, g_n\}$ *un ensemble de fonctions booléennes détectées dans la formule originale telle que les différents ensembles de variables d'entrée associées aux différentes fonctions booléennes dans G soient tous disjoints et tous inclus dans* α. *Soit* $\sigma_1, \ldots, \sigma_n$ *la séquence de clauses apprises obtenues successivement après chaque substitution t.q.à l'étape i,* σ_i *est obtenue en remplaçant un ensemble de variables d'entrée dans* σ_{i-1} *par une variable de sortie* y_j, *alors* $|\sigma_i| < |\sigma_{i-1}|$, *et* $|\sigma_i| > 1$, $1 \leq i, j \leq n$

Preuve 2 *Soit* σ_0 *la clause initialement apprise. Comme* σ_i *est obtenue par substitution d'au moins deux littéraux (entrée d'une fonction booléenne) dans* σ_{i-1} *par un littéral qui est la sortie de cette fonction, on a* $|\sigma_i| < |\sigma_{i-1}|$, *pour* $1 \leq i \leq n$. *De plus,* $\forall 1 < i < n$, $x \in \sigma_i$, *où x est le littéral assertif et n'est pas considéré dans la substitution, alors* $|\sigma_i| > 1$, $1 \leq i \leq n$

Propriété 3 *Etant donné une clause apprise* σ, *et un ensemble* $\mathcal{G} = \{g_1, \ldots, g_n\}$ *de fonctions booléennes détectées dans la formule CNF originale. La complexité temporelle de notre approche de substitution dynamique est quadratique dans le pire des cas.*

Preuve 3 *On a* $|\mathcal{G}|$ *fonctions booléennes. Pour chaque fonction booléenne, on fait* $|\sigma|$ *comparaisons. Ainsi, on peut faire au plus* $|\mathcal{G}| \times |\sigma|$ *comparaisons pour réaliser la substitution.*

4 Expérimentations

Les expérimentations menées sont effectuées sur un large panel d'instances industrielles et crafted provenant de la compétition SAT 2009 et de la SAT-Race 2009. Toutes les instances sont simplifiées par prétraitement `SatElite` [12]. On peut noter que `SatElite` substitue certaines fonctions booléennes à l'étape de pré-traitement. Par conséquent, les fonctions booléennes détectées sont celles qui restent dans la formule simplifiée par `SatElite`. Nous avons intégré notre approche de substitution dynamique dans `Minisat` 2.2 [13] et nous avons effectué une comparaison entre les résultats obtenus par les solveurs originaux et ceux incluant l'approche de substitution dynamique. Il est aussi important de noter le détail d'implémentation suivant : Quand les variables d'entrée sont substituées par la variable de sortie, l'activité de la variable de sortie est mise à jour de la même manière que les autres variables de la clause apprise. Ceci permet de focaliser la recherche sur certaines variables de sortie. Ce détail d'implémentation est similaire à celui implémenté dans [3].

Tous les tests sont effectués sur un cluster Xeon 3.2GHz (2 GB RAM). Les résultats du temps de calcul sont indiqués en secondes.
Pour ces expérimentations, le temps de calcul limite est fixé à 1 heure. les différentes tables montrent un ensemble représentatif de familles d'instances.

4.1 Problèmes industriels

Les tables 1 et 2 détaillent les résultats sur les problèmes SAT industriels issus de la compétition SAT'2009 et SAT-Race'2010 en utilisant respectivement les méthodes de détection syntaxique et sémantique pour la détection des fonctions booléennes dans la formule originale.

Pour chaque instance, nous reportons son nom (Instance), son nombre de variables et de clauses (#Var, #Cl), le temps utilisé par `Minisat` pour résoudre l'instance, le temps utilisé par `Minisat+DS` pour résoudre l'instance, le nombre total de substitution éffectuée ($nbSub$) et le nombre total de littéraux éliminés par la substitution des entrées par les sorties ($totalSub$).

La table 1 représente un focus sur quelques familles industrielles. Sur ces familles, les améliorations sont relativement importantes. Par exemple, si on considère la famille vmpc, on peut voir que notre substitution dynamique permet d'avoir un gain d'un ordre de magnitude (instances 24 and 27). Sur la même famille, `Minisat+DS` résoud 2 instances de plus que `Minisat` (instances 31 and 33).
Sur la famille gss, sur les 7 instances résolus par `Minisat+DS` et `Minisat`, `Minisat+DS` est meilleur sur 5 instances. Sur la même famille, `Minisat+DS` résoud une instance de plus que `Minisat` (instance 23).

En résumé, on peut voir que sur certaines familles, `Minisat+DS` est plus rapide et résoud plus de problèmes que `Minisat`.

4.2 Problèmes crafted

Ces problèmes sont conçus à la main et beaucoup d'entre eux sont conçus dans le but de battre tous les solveurs DPLL existants. Ils contiennent par exemple les instances Quasi-group, instances SAT forcés aléatoires, counting, instances "ordering" et "pebbling", problèmes sociaux du golfeur, etc.

La table 3 montre que sur les 11 instances résolues de la famillle QG, `Minisat+DS` résoud plus efficacement

7 instances. Sur la famille rbsat, `Minisat` est meilleur sur 5 instances parmi les 8 instances résolues. On peut remarquer que sur 4 de ces 5 instances, la valeur de nbSum et totalSub est égal à 0 ce qui peut expliquer les limites de l'efficacité de notre approche dans ces cas.

Globalement, on peut voir que l'ajout de notre processus de substitution dynamique à `Minisat` améliore ses performances sur certaines familles de problèmes crafted.

La table 4 montre que ces performances sont encore améliorées en utilisant la méthode sémantique pour la détection des fonctions boolénnes. Dans cette table, sur la famille QG, les temps de résolution de `Minisat+DS` ont été améliorés de manière significative.

4.3 Summary

Dans l'ensemble, nos expérimentations nous ont permis de démontrer deux choses. Premièrement, notre technique ne dégrade pas mais au contraire améliore souvent les performances des solveurs DPLL sur les problèmes industriels et crafted. Second, elle améliore l'applicabilité de ces algorithmes sur des classes de problèmes qui sont faites pour être difficiles pour eux. Les résultats présentés dans les tables précédentes montre que notre approche est éfficace sur certaines familles d'instances SAT. Globalement, sur l'ensemble d'instances, notre approche est compétitive et résoud un nombre similaire d'instances.

A partir de ces expérimentations, on observe que notre méthode présente certaines complémentarités avec les solveurs SAT classique tels que `Minisat`. En effet, on observe que plusieurs instances sont résolues uniquement par notre approche tandis que d'autres ne sont résolues que par `Minisat`. Sur de nombreux cas où notre approche dégrade, le nombre de substitution est assez faible.

5 Remerciements

Ce travail a été soutenu par l'Agence Nationale de la Recherche - projet ANR programme blanc TUPLES.

6 Conclusion

Ce papier présente une nouvelle technique de substitution dynamique de fonctions booléennes détectées dans une formule booléenne sous forme CNF. Cette approche peut être vue comme une façon originale de manipuler les sous-formules représentées par ces fonctions. Elle nous permet d'imiter le principe de résolution étendue. En effet, notre approche exploite les fonctions booléennes cachées généralement introduites pendant la phase d'encodage CNF, et les utilise pour minimiser les clauses apprises. Ce qui nous permet d'exploiter ces variables auxiliaires au lieu d'en ajouter des variables supplémentaires au cours de la phase de résolution. La substitution des variables d'entrée par les variables de sortie au cours de l'analyse des conflits, est une façon élégante de manipuler ces variables de sortie. Les résultats expérimentaux sur certaines classes d'instances SAT, montrent clairement que notre approche est compétitive avec l'état de l'art des solveurs SAT, bien qu'il présente certains aspects de complémentarité. De plus, notre approche présente des améliorations intéressantes sur certaines familles d'instances SAT.

Comme travaux futures, nous envisageons d'étendre notre approche en visant une substitution partielle. En effet, même si l'ensemble des variables d'entrée apparaît partiellement dans la clause apprise donnée, on peut toujours substituer ces variables par la variable de sortie correspondante. Dans un tel cas, il pourrait être intéressant de calculer la substitution qui maximise la réduction de la taille de la clause apprise.

Références

[1] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *Transactions on Computer Aided Design*, 2003.

[2] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Saïs. Generalized framework for conflict analysis. In *Proceedings of the eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, pages 21–27, 2008.

[3] Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

[4] Fahiem Bacchus. Enhancing davis putnam with extended binary clause reasoning. In *AAAI/IAAI*, pages 613–619, 2002.

[5] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *SAT*, pages 341–355, 2003.

[6] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.

Instance	(#Var, #Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
vmpc_24	(576 , 67872)	SAT	18.41	**4.49**	9	198
vmpc_25	(625 , 76775)	SAT	**19.89**	57.37	134	3082
vmpc_26	(676 , 86424)	SAT	**14.31**	76.46	267	6408
vmpc_27	(729 , 96849)	SAT	43.66	**10.06**	24	600
vmpc_28	(784 , 108080)	SAT	96.72	**62.55**	189	4914
vmpc_29	(841 , 120147)	SAT	**70.34**	755.24	1220	32994
vmpc_30	(900 , 133080)	SAT	**476.82**	877.05	1223	34244
vmpc_31	(961 , 146909)	SAT	timeout	**3366.28**	2915	84564
vmpc_33	(1089 , 177375)	SAT	timeout	**3151.05**	2340	72540
vmpc_34	(1156 194072)	SAT	**timeout**	**timeout**	2048	65568
gss-16-s100	(31248 , 93904)	SAT	18.04	**6.15**	986	1132
gss-20-s100	(31503 , 94748)	SAT	1275.94	**7.22**	646	736
gss-23-s100	(31711 , 95400)	SAT	timeout	**2213.94**	118104	136100
gss-17-s100	(31318 , 94116)	SAT	107.71	**40.26**	3706	4949
gss-19-s100	(31435 , 94548)	SAT	152.60	**105.78**	7533	8797
gss-13-s100	(30867 , 92735)	SAT	6.21	**5.78**	366	466
gss-14-s100	(31229 , 93855)	SAT	**4.05**	11.63	612	655
gss-15-s100	(31238 , 93878)	SAT	**12.76**	30.73	3257	4336
sha0_35_1	(48689 , 204053)	SAT	71.23	**23.12**	7421	10241
sha0_36_5	(50073 , 210223)	SAT	471.32	**17.51**	4318	5498
sha0_35_3	(48689 , 204067)	SAT	29.02	**9.78**	1617	1879
sha0_35_2	(48689 , 204071)	SAT	**22.88**	202.25	83757	110310
UTI-20-5p0	(225296 1192799)	UNSAT	timeout	**1705.26**	326191	2062182
rbcl_xits_07	(1128 , 57446)	UNSAT	219.12	**156.02**	112280	372012
uts-l06-ipc5-h35	(175670 837466)	SAT	3.20	**0.84**	5	10
md5_48_1	(66892 , 279248)	SAT	221.94	**145.82**	26021	34469
md5_47_4	(65604 , 273506)	SAT	64.13	**21.60**	3839	4614
md5_48_3.cnf	(66892 , 279258)	SAT	**160.06**	210.04	42255	57303
partial-10-15-s	(261020 , 1211106)	SAT	3039.08	**649.65**	13696	18345
partial-10-13-s	(234673 , 1071339)	SAT	timeout	**1470.39**	31336	43086
rpoc_xits_08	(1278 , 74789)	UNSAT	timeout	**2974.28**	459189	1528680
uts-l06-ipc5-h32	(163755 , 800163)	UNSAT	**10.62**	48.07	392	28040
maxxororand032	(23696 , 70703)	UNSAT	859.81	**542.29**	241743	1051278
manol-pipe-g10bidw	(237485 , 705220)	UNSAT	494.72	**172.32**	40268	75273
manol-pipe-c7nidw	(169072 , 501421)	UNSAT	**33.35**	54.45	39464	60577
simon-s02b-dp11u10	(9197 , 25271)	UNSAT	338.53	**295.11**	350707	421961
mizh-sha0-36-2	(50073 , 210239)	SAT	892.01	**27.77**	8888	11784
mizh-sha0-36-3	(50073 , 210235)	SAT	**110.29**	502.11	239539	341651
mizh-md5-48-5	(66892 , 279256)	SAT	74.91	**10.83**	2523	2902
velev-pipe-sat-1.0-b10	(118040 , 8804672)	SAT	**58.25**	67.40	0	0

TABLE 1 – Instances Industrielles : substitution dynamique et détection par la méthode syntaxique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
gss-17-s100	(31318 , 94116)	SAT	107.71	**96.04**	6440	7456
gss-20-s100	(31503 , 94748)	SAT	1275.94	**425.667**	28295	35016
sha0_35_1	(48689 , 204053)	SAT	71.23	**27.67**	9182	12319
sha0_36_5	(50073 , 210223)	SAT	471.32	**130.26**	50706	69396
md5_48_1	(66892 , 279248)	SAT	221.94	**65.12**	11265	14152
c8idw_s	(122321 , 361795)	UNSAT	6.18	**0.41**	239	321
f7nidw	(310434 , 923497)	UNSAT	1867.02	**1217.44**	41689	53540
g7nidw	(75496 , 222379)	UNSAT	62.36	**17.54**	3059	4862
eq.atree.braun.11	(1694 , 5726)	UNSAT	3593.43	**3472.79**	20198398	32717812
rbcl_xits_07	(1128 , 57446)	UNSAT	219.12	**169.61**	104596	346524
rpoc_xits_07	(1128 , 63345)	UNSAT	197.91	**185.48**	86856	282885
gus-md5-09	(69487 , 226581)	UNSAT	204.46	**193.52**	12015	20529
mizh-sha0-36-2	(50073 , 210239)	SAT	892.01	**272.56**	97731	135321
simon-s02b-dp11u10	(9197 , 25271)	UNSAT	338.53	**258.07**	280825	331549
manol-pipe-g7nidw	(75496 , 222379)	UNSAT	62.64	**15.94**	3059	4862
mizh-sha0-36-3	(50073 , 210235)	SAT	110.29	**25.71**	7870	10689
mizh-md5-47-3	(65604 , 273522)	SAT	117.49	**39.97**	8184	10314
schup-l2s-motst-2	(507145 , 1601920)	SAT	11.75	**9.26**	85	246
post-cbmc-aes-ee	(498723 , 2928183)	UNSAT	568.25	**489.90**	429	2173
ndhf_xits_21_SAT	(4466 , 542457)	SAT	1.41	**0.55**	618	16716
post-cbmc-aes-ele	(270963 , 1601376)	UNSAT	7.45	**4.84**	22	66
dated-5-15-u	(151952 , 697321)	UNSAT	**323.40**	418.72	6085	11012
dated-10-13-s	(181082 , 824258)	SAT	**5.69**	20.79	92	149
q_query_3_l48_lambda	(34656 , 174528)	UNSAT	**78.88**	141.38	164871	195089
q_query_3_l45_lambda	(32313 , 161529)	UNSAT	**91.85**	162.14	182561	216837
zfcp-2.8-u2-nh	(10950109, 32697150)	SAT	**27.73**	31.30	0	0
vmpc_25	(625 , 76775)	SAT	**19.89**	49.05	73	1613
mizh-sha0-35-3	(48689 , 204067)	SAT	**28.49**	36.87	11529	18419
mizh-sha0-36-4	(50073 , 210235)	SAT	**89.19**	111.12	44991	64725
goldb-heqc-term1mul	(3504 , 22229)	UNSAT	**42.02**	69.53	17	51
countbitssrl032	(18607 , 55724)	UNSAT	**646.13**	903.15	135175	171439

TABLE 2 – Instances Industrielles : substitution dynamique et détection par la méthode sémantique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
QG7a-gensys-icl004	(2401 , 15960)	UNSAT	523.70	**401.41**	672102	3433306
QG6-gensys-icl001	(1587 , 8013)	UNSAT	245.77	**218.43**	856139	4361401
QG6-gensys-icl004	(1605 , 8025)	UNSAT	189.05	**93.22**	454510	2154537
QG8-gensys-ukn005	(1133 , 56210)	UNSAT	51.20	**35.32**	10267	70734
QG7a-gensys-ukn005	(2765 , 18046)	SAT	47.53	**26.45**	55066	313842
QG-gensys-icl003	(1472 , 7737)	UNSAT	134.79	**87.20**	409490	1884892
QG7a-gensys-ukn001	(2737 , 18375)	SAT	42.38	**9.49**	23785	137159
QG7-gensys-icl006	(728 , 17199)	UNSAT	**1311.19**	1483.74	292722	1388841
QG7-dead-dnd005	(502 , 11816)	UNSAT	**431.93**	696.91	220929	1119611
QG6-dead-dnd002	(1339 , 7095)	UNSAT	**319.51**	418.85	1574081	8255326
QG-gensys-brn008	(1467 , 752)	UNSAT	**98.60**	105.40	487138	2224629
rbsat-v760c43649g8	(760 , 43649)	SAT	39.20	**29.59**	6	114
rbsat-v760c43649gyes4	(760 , 43649)	SAT	7.29	**3.30**	1	23
rbsat-v760c43649gyes6	(760 ,43649)	SAT	754.22	**118.22**	241	4338
rbsat-v760c43649gyes1	(760 , 43649)	SAT	**2481.48**	2499.2	0	0
rbsat-v760c43649g4	(760 , 43649)	UNSAT	**546.31**	622.815	0	0
rbsat-v760c43649g1	(760 , 43649)	SAT	**178.13**	197.923	0	0
rbsat-v760c43649g10	(760 , 43649)	SAT	**68.83**	121.836	78	1658
rbsat-v945c61409gyes9	(945 , 61409)	SAT	**824.15**	831.814	0	0
rbsat-v945c61409gyes4	(945 , 61409)	SAT	timeout	**3453.15**	241	5613
mod3block_2vars_11	(526 , 146535)	SAT	timeout	**1488.51**	340924	355550
mod4block_2vars_10	(479 , 123509)	SAT	2393.86	**679.48**	12822	26072
em_7_4_8_all	(1617 , 37237)	SAT	134.58	**8.49**	1016	6362
em_8_4_5_all	(2420 , 62900)	SAT	**32.08**	52.32	16490	61304
em_11_3_4_all	(8713 , 414651)	SAT	**14.31**	87.01	5785	15145
em_7_4_8_exp	(1617 , 25837)	SAT	194.67	**43.15**	2822	19580
em_7_3_6_exp	(1473 , 22887)	SAT	**2.46**	17.86	4007	17174
em_7_3_6_fbc	(1473 , 19063)	SAT	1702.68	**101.68**	42879	186028
em_8_4_5_fbc	(2420 , 33572)	SAT	timeout	**1012.14**	445179	1547530
em_7_3_6_cmp	(1473 , 11563)	SAT	761.50	**299.31**	72641	245066
em_8_4_5_cmp	(2420 , 22340)	SAT	1127.77	**782.49**	355928	1144871
em_7_4_8_cmp	(1617 , 14513)	SAT	**2783.97**	timeout	621973	2334680
instance_n9_i9_pp_ci_ce	(33867 , 457280)	SAT	timeout	**1482.43**	519105	907405
instance_n9_i9_pp	(33867 , 454832)	SAT	640.35	**339.55**	144223	234487
instance_n5_i6_pp_ci_ce	(4380 , 31980)	UNSAT	**1.27**	1.58	700	1621
instance_n8_i8_pp_ci_ce	(21640 , 257551)	SAT	523.53	**253.46**	128221	213622
strips-gripper-18t35	(11318 , 316793)	SAT	36.08	**16.15**	16	16

TABLE 3 – Instances crafted : substitution dynamique et détection par la méthode syntaxique

Instance	(#Var,#Cl)	Statut	Minisat	Minisat + DS		
			time(s)	time(s)	nbSub	totalSub
QG7a-gensys-icl004	(2401 , 15960)	UNSAT	523.70	**352.33**	672102	3433306
QG6-gensys-ukn003	(2123 , 9177)	UNSAT	492.87	**448.36**	1249830	6229729
rbsat-v945c61409gyes9	(945 , 61409)	SAT	824.15	**774.67**	0	0
QG6-gensys-brn007	(1477 , 7809)	UNSAT	119.94	**103.15**	444274	2185145
QG7-gensys-icl006	(728 , 17199)	UNSAT	**1311.19**	1447.36	292722	1388841
QG7-dead-dnd005	(502 , 11816)	UNSAT	**431.93**	682.53	220929	1119611
QG6-gensys-icl001	(1587 , 8013)	UNSAT	245.77	**215.35**	856139	4361401
QG6-gensys-icl004	(1605 , 8025)	UNSAT	189.05	**91.66**	454510	2154537
QG8-gensys-ukn005	(1133 , 56210)	UNSAT	51.20	**30.95**	10267	70734
QG7a-gensys-ukn005	(2765 , 18046)	SAT	47.53	**26.39**	55066	313842
QG-gensys-icl003	(1472 , 7737)	UNSAT	134.79	**85.74**	409490	1884892
QG7a-gensys-ukn001	(2737 , 18375)	SAT	42.38	**9.35**	23785	137159
mod4block_2vars_11	(530 , 153478)	SAT	629.84	**120.62**	7769	15662
mod3block_2vars_11	(526 , 146535)	SAT	timeout	**1362.28**	340924	355550
mod3_4vars_6gates	(289 , 33900)	UNSAT	**517.72**	540.37	7704	11264
mod4block_2vars_10	(479 , 123509)	SAT	2393.86	**638.17**	12822	26072
rbsat-v760c43649g8	(760 , 43649)	SAT	39.20	**28.33**	6	114
rbsat-v760c43649g10	(760 , 43649)	SAT	**68.83**	116.14	78	1658
rbsat-v760c43649gyes4	(760 , 43649)	SAT	7.29	**3.26**	1	23
rbsat-v760c43649gyes6	(760 , 43649)	SAT	754.22	**111.77**	241	4338
rbsat-v760c43649gyes1	(760 , 43649)	SAT	**2481.48**	346.825	52	1144
rbsat-v945c61409gyes4	(945 , 61409)	SAT	timeout	**3217.75**	241	5613
rbsat-v760c43649gyes1	(760 , 43649)	SAT	2481.48	2363.94	0	0
em_7_4_8_all	(1617 , 37237)	SAT	134.58	**8.49**	1016	6362
em_7_4_8_exp	(1617 , 25837)	SAT	194.67	**41.18**	2822	19580
em_8_4_5_all	(2420 , 62900)	SAT	**32.08**	50.10	16490	61304
em_11_3_4_all	(8713 , 414651)	SAT	**14.31**	86.03	5785	15145
em_12_2_4_all	(12584 , 729136)	SAT	**24.86**	664.82	30255	79505
em_7_3_6_fbc	(1473 , 19063)	SAT	1702.68	**97.85**	42879	186028
em_8_4_5_fbc	(2420 , 33572)	SAT	timeout	**953.59**	445179	1547530
em_7_3_6_cmp	(1473 , 11563)	SAT	761.50	**290.91**	72641	245066
em_9_3_5_exp	(3857 , 108164)	SAT	**42.82**	376.20	140442	517796
em_8_4_5_cmp	(2420 , 22340)	SAT	1127.77	**725.73**	355928	1144871
instance_n9_i9_pp_ci_ce	(33867 , 457280)	SAT	timeout	**1476.45**	519105	907405
instance_n9_i9_pp	(33867 , 454832)	SAT	640.35	**335.01**	144223	234487
instance_n8_i8_pp_ci_ce	(21640 , 257551)	SAT	523.53	**235.77**	128221	213622
sgen1-unsat-85-100	(85 , 180)	UNSAT	**499.61**	596.44	0	0
sgen1-unsat-73-100	(73 , 156)	UNSAT	**38.53**	51.36	0	0
sgen1-unsat-97-100	(97 , 204)	UNSAT	**3561.91**	timeout	0	0
sgen1-sat-160-100	(160, 384)	SAT	**44.29**	55.17	0	0
strips-gripper-18t35	(11318 , 316793)	SAT	36.08	**14.53**	16	16
999999000001nw	(4667 , 18492)	UNSAT	1287.23	**1082.17**	12	12

TABLE 4 – Instances Crafted : substitution dynamique et détection par la méthode sémantique

[7] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1194–1201, 2003.

[8] Belaid Benhamou and Lakhdar Saïs. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1) :89–102, February 1994.

[9] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8 :28–32, October 1976.

[10] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159. 1996.

[11] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[12] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75, 2005.

[13] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2002.

[14] Pierre Flener, Justin Pearson, Meinolf Sellmann, Pascal Van Hentenryck, and Magnus Ågren. Dynamic structural symmetry breaking for constraint satisfaction problems. *Constraints*, 14(4) :506–538, 2009.

[15] Ian P. Gent and Barbara Smith. Symmetry breaking in constraint programming. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence(ECAI'00)*, pages 599–603, 2000.

[16] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, 1998.

[17] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 122–132, 2004.

[18] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297–308, 1985.

[19] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI'02)*, pages 674–682, 2002.

[20] Balakrishnan Krishnamurthy. Shorts proofs for tricky formulas. *Acta Informatica*, 22 :253–275, 1985.

[21] Balakrishnan Krishnamurthy and Robert N. Moll. Examples of hard tautologies in the propositional calculus. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing (STOC'81)*, pages 28–37, 1981.

[22] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[23] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[24] R. Ostrowski, É. Grégoire, B. Mazure, and L. Saïs. Recovering and exploiting structural knowledge from cnf formulas. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming(CP'02)*, pages 185–199, Ithaca (N.Y.), 2002.

[25] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers with restarts. In *CP*, pages 654–668, 2009.

[26] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, pages 350–361, 1993.

[27] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.