Subsumption dirigée par l'analyse de conflits

Youssef Hamadi, Said Jabbour, Lakhdar Saïs

Subsumption dirigée par l'analyse de conflits

Youssef Hamadi[1] **Said Jabbour**[2] **Lakhdar Sais**[2]

[1] Microsoft Research
7 J J Thomson Avenue
Cambridge, United Kingdom
youssefh@microsoft.com

[2] CRIL-CNRS, Université d'Artois
rue Jean Souvraz SP18
F-62307 Lens Cedex France
{jabbour,sais}@cril.fr

Abstract

Cet travail présente une technique originale de subsumption pour les formules booléennes sous forme normale conjonctive (CNF). Il exploite une condition simple et suffisante pour détecter, durant l'analyse de conflits, les clauses de la formule qui peuvent être réduites par subsumption. Durant le processus de dérivation de la clause à apprendre, et à chaque étape du processus de résolution associé, le test de subsumption entre la résolvante courante et une clause de la formule originale ou apprise auparavant est alors efficacement effectué. La méthode résultante permet d'éliminer dynamiquement des littéraux appartenant aux clauses de la formule courante. Les résultats expérimentaux montrent que l'intégration de notre technique de subsumption dynamique dans les solveurs `Minisat` et `Rsat` est bénéfique, et cela particulièrement sur les problèmes dits "crafted".

1 Introduction

Le problème SAT, i.e., la vérification de la satisfaction 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 ainsi que par la 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 [13, 9], sont basés sur la propagation de contraintes classique [6] efficacement combinée à d'efficaces structures de données avec : (i) stratégies de redémarrage [10, 11], (ii) heuristiques de choix de variables (comme VSIDS) [13], et (iii) apprentissage de clauses [1, 12, 13]. Les solveurs SAT modernes peuvent être vus comme une extension de la procédure DPLL classique obtenues grâce à différentes améliorations. Il est important de noter que la règle de résolution basique à la Robinson joue encore un rôle prépondérant dans l'efficacité des solveurs modernes qui peut être vu comme une forme particulière de la résolution générale [2].

En effet, l'apprentissage basé sur les conflits, l'un des composants importants des solveurs SAT est basé sur la résolution. On peut mentionner aussi que l'un des prétraitement le plus connu et dont le succès est largement répandu (`SatElite`) est basé sur l'élimination de variables par l'application de la règle de résolution [16, 3]. Comme cité dans [16], sur les instances industrielles, la résolution génère plusieurs résolvantes tautologiques. Ceci peut être expliqué par le faite que plusieurs clauses codant des fonctions booléennes sont reliées par un ensemble de variables communes. Cette propriété du codage peut être à l'origine de plusieurs clauses redondantes ou subsumées aux difèrentes étapes du processus de recherche.

L'utilité de (`SatElite`) sur les problèmes industriels étant prouvée, l'on est en droit de se demander si l'application de la règle de résolution pourrait être réalisée non seulement comme une étape de pré-traitement mais systématiquement, pendant le processus de recherche. Malheureusement, maintenir une formule fermée par subsumption peut être très coûteux. Une tentative a été faite récemment dans cette direction par L. Zhang [17]. Dans ce travail, un nouvel algorithme maintient la base de clauses fermée par subsumption en supprimant dynamiquement les clauses subsumées au moment de leur ajout. Plus intéressant, l'auteur indique cette perspective de recherche : "trouver un équilibre entre le coût d'exécution et la qualité de la simplification par subsumption à la volée d'une CNF est un problème qui mérite plus d'intérêt et d'investigation".

Dans cet article, notre objectif est de concevoir un algorithme de subsumption dynamique simple et efficace basé sur la résolution. L'approche proposée a pour but d'éliminer des littéraux de la formule CNF en substituant dynamiquement les clauses originales par des clauses plus courtes. Plus précisemment, nous exploitons les étapes intermidiares de l'analyse de conflits classique pour subsumer des clauses de la formule qui sont utilisées dans la processus de résolution relatif à la génération de la clause assertive. Comme les clauses originales ou les clauses aprisent peuvent être utilisées durant l'analyse de conflits les deux catégories peuvent être simplifiées.

L'efficacité de notre technique réside dans la simplicité du test de subsumption, qui est basé sur une condition simple et suffisante calculable en temps constant. En plus, du faite que notre technique repose sur le processus de génération de la clause conflit, elle est aussi guidée par les conflits, et simplifie la partie de la formule identifiée comme étant importante par la stratégie de recherche (dirigé par VSIDS). Ce processus dynamique préserve la satisfiabilité de la formule, et avec quelques règles de compatibilité additionnelles peut préserver l'équivalence des modèles.

Notre article est organisé comme suit. Après quelques notations et définitions préliminaires, le graphe d'implications classique et le schéma d'apprentissage sont présentés dans la section 2. Ensuite notre approche de subsumption dynamique est décrite dans la section 3. Finalement, avant la conclusion, des résultats expérimentaux démontrent les performances de notre approche sur les catégories d'instances testées.

2 Definitions

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 positif (x) ou negatif ($\neg x$) propositional variable. Les deux littéraux x et $\neg x$ sont appelés *complémentaire*. On note par $\bar{l}$ le littéral complémentaire de l. Pour un ensemble de littéraux L, $\bar{L}$ est définit 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 exactement deux littéraux. Une *clause vide*, notée $\bot$, est interprété comme fausse (unsatisfiable), 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 contient pas de littéraux complémentaites. Une *affectation* ρ d'une formule booléenne $\mathcal{F}$ est une fonction qui associe la valeur $\rho(x) \in \{vrai, faux\}$ 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 fondamentale, de façon évidente un *modèle* d'une formule $\mathcal{F}$ est une affectation ρ qui rend la formule $vraie$; notée $\rho \vDash \mathcal{F}$.

Les notations suivantes sont largement utilisées le reste de l'article.

- $\eta[x, c_i, c_j]$ dénote la *résolvante* entre une clause c_i contenant un littéral x et c_j une autre clause contenant l'opposé de ce même littéral $\neg x$.
  En d'autres termes, $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. Une résolvante est appelée *tautologique* s'elle contient à la fois un littéral et son opposé.
- $\mathcal{F}|_x$ représente la formule obtenu à partir de $\mathcal{F}$ en affectant à x la value de vérité *vrai*. Formellement $\mathcal{F}|_x = \{c \mid c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \mathcal{F}, \neg x \in c\}$ (les clauses contenant x (satisfaites) sont supprimées ; et celles contenant $\neg x$ sont simplifiées). Cette notation est étendue aux affectations : étant donnée une affectation $\rho = \{x_1, \ldots, x_n\}$, on définit $\mathcal{F}|_\rho = (\ldots((\mathcal{F}|_{x_1})|_{x_2})\ldots|_{x_n})$.
- $\mathcal{F}^*$ dénote la formule $\mathcal{F}$ fermée par propagation unitaire, définit récursivement comme suit : (1) $\mathcal{F}^* = \mathcal{F}$ si $\mathcal{F}$ ne contient pas des clauses unitaires, (2) $\mathcal{F}^* = \bot$ si $\mathcal{F}$ contient deux clauses unitaires $\{x\}$ et $\{\neg x\}$, (3) sinon, $\mathcal{F}^* = (\mathcal{F}|_x)^*$ tel que x est le littéral apparaissant dans une clause unitaire de $\mathcal{F}$. Une clause c est déduite par propagation unitaire à partir de $\mathcal{F}$, notée $\mathcal{F} \models^* c$, si $(\mathcal{F}|_{\bar{c}})^* = \bot$.

Soient c_1 et c_2 deux clauses d'une formule $\mathcal{F}$. On dit que c_1 (respectivement c_2) subsume (respectivement est subsumée) c_2 (respectivement par c_1) ssi $c_1 \subseteq c_2$. Si c_1 subsume c_2, alors $c_1 \models c_2$ (l'inverse est faux). Aussi $\mathcal{F}$ et $\mathcal{F} - c_2$ sont équivalentes pour la satisfiabilité.

2.2 Recherche DPLL

Nous introduisons maintenant des notations et remarques terminologiques associées aux solveurs SAT basés sur la procédure de Davis Logemann Loveland, communément appelé DPLL [6]. DPLL 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 consèquence (égal au niveau du backtrack). Au niveau i, l'interprétation partielle courante ρ peut être représentée comme une séquence décision–propagations de la forme $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ telle que le premier lit-

téral 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 litéraux unitaires propagés à ce même niveau i.

Soit $x \in \rho$, On note $l(x)$ le niveau d'affectation de x, $d(\rho, i) = x$ si x est le littéral de décision affecté au niveau i.

2.3 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 mdernes. À chaque fois qu'un littéral y est propagé, on sauvegarde une référence de la clause impliquant la propagation de y, qu'on note $\overrightarrow{cla}(y)$. La clause $\overrightarrow{cla}(y)$, 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 ($\rho(x_i) = \textit{faux}, \forall i \in 1..n$), alors que $\rho(y) = \textit{vrai}$. Lorsqu'un littéral y n'est pas obtenu par propagation mais issue d'une décision, $\overrightarrow{cla}(y)$ est indéfinie, qu'on note par convention $\overrightarrow{cla}(y) = \perp$. Quand $\overrightarrow{cla}(y) \neq \perp$, on note par $exp(y)$ l'ensemble $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, appelé ensemble des *explications* de y. Alors que $\overrightarrow{cla}(y)$ 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 (propagé 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, exp est éliminée, et un graphe d'impliacations est tout simplement noté $\mathcal{G}_{\mathcal{F}}^{\rho}$. On note par m le niveau du conflit.

Exemple 1 *$\mathcal{G}_{\mathcal{F}}^{\rho}$, montré dans la Figure 1 représente le graphe d'implications de la formule $\mathcal{F}$ et l'affectation partiel 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 $\rho(\mathcal{F}) = faux$.

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

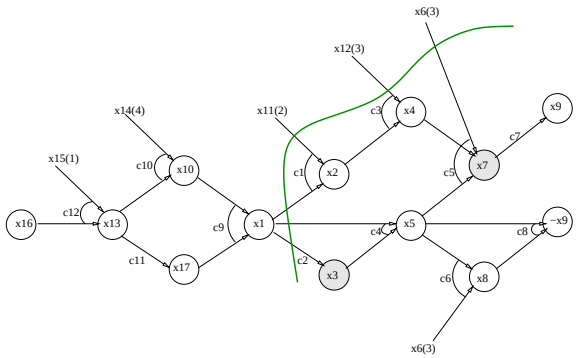

FIG. 1 – Graphe d'implications $\mathcal{G}_{\mathcal{F}}^{\rho} = (\mathcal{N}, \mathcal{E})$

L'analyse du conflit est le résultat de l'application de la résolution à partir de la clause conflit en utilisant différentes implications implicitement codées dans le graphe d'implications. On appelle ce processus dérivation de la clause assertive (DCA en court).

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

1. *$\sigma_1 = \eta[x, \overrightarrow{cla}(x), \overrightarrow{cla}(\neg x)]$, tel que $\{x, \neg x\}$ est le conflit.*
2. *σ_i, pour $i \in 2..k$, est construite en sélectionnant un littéral $y \in \sigma_{i-1}$ pour lequel $\overrightarrow{cla}(\overline{y})$ est définit. Nous avons alors $y \in \sigma_{i-1}$ et $\overline{y} \in \overrightarrow{cla}(\overline{y})$: appliquer la résolution entre ces deux clauses mène à la clause σ_i qui est définit comme $\eta[y, \sigma_{i-1}, \overrightarrow{cla}(\overline{y})]$;*
3. *σ_k est en plus une clause assertive.*

Notons que chaque σ_i est une résolvante de la formule $\mathcal{F}$: par induction, σ_1 est la résolvante entre deux clauses qui appartiennent directement à $\mathcal{F}$; pour chaque $i > 1$, σ_i est une résolvante entre σ_{i-1} (qui est, par l'hypothèse d'induction, une résolvante) et une clause de $\mathcal{F}$. Chaque σ_i est également un *impliqué* de $\mathcal{F}$, ce qui veut dire que : $\mathcal{F} \models \sigma_i$. Par définition du graphe d'implication, on a aussi $\mathcal{F}' \models^* \sigma_i$ tel que $\mathcal{F}' \subset \mathcal{F}$ est l'ensemble des clauses utilisées pour générer σ_i.

Considérons à nouveau l'exemple 1. Le parcours du graphe $\mathcal{G}_{\mathcal{F}}^{\rho}$ (voir Fig. 1) est décrit par la dérivation de la clause assertive suivant :

$\langle \sigma_1, \ldots, \sigma_7 \rangle$ tel que $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_5}^5 \vee \neg {x_7}^5 \vee \neg {x_8}^5)$ et $\sigma_7 = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$. Le nœud x_1 correspond au littéral assertif, $\neg x_1$ est appelé le premier Unique Point d'Implication (First UIP).

3 Apprentissage et subsumption dynamique

Dans cette section, nous montrons comment l'apprentissage classique peut être adapté pour une subsumption dynamique et efficace des clauses. Dans notre approche, on exploite les différentes clauses générées pendant les étapes de résolution du processus d'analyse de conflit. Bien sûr, il est possible de considérer la subsumption entre chaque résolvante générée et l'ensemble entier des clauses. Cependant, celà peut être coûteux en pratique. Illustrons certaines des principales caractéristiques de notre approche proposée.

3.1 Exemple

Considérons à nouveau l'exemple 1 et le graphe d'implications $\mathcal{G}^{\rho}_{\mathcal{F}}$ (figure 1). Le dérivation de la clause assertive permettant de déduire la clause Δ_1 est décrit comme suit :
$\pi = \langle \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_7 = \Delta_1 \rangle$
- $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_8}^5 \vee \neg {x_7}^5 \vee \neg {x_5}^5)$
- $\sigma_2 = \eta[x_8, \sigma_1, c_6] = (\neg {x_6}^3 \vee \neg {x_7}^5 \vee \neg {x_5}^5)$
- $\sigma_3 = \eta[x_7, \sigma_2, c_5] = (\neg {x_6}^3 \vee \neg {x_5}^5 \vee \neg {x_4}^5) \subset c_5$ (subsumption)
- $\ldots$
- $\Delta_1 = \sigma_7 = \eta[x_2, \sigma_6, c_1] = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$

Comme on peut le voir, ce processus π inclut la résolvante $\sigma_3 = (\neg {x_6}^3 \vee \neg {x_5}^5 \vee \neg {x_4}^5)$ qui subsumes la clause $c_5 = (\neg x_6 \vee \neg x_5 \vee \neg x_4 \vee x_7)$. Par conséquent, le littéral x_7 peut être éliminé de la clause c_5. En général, la résolvante σ_3 peut subsumer d'autres clauses du graphe d'implications qui contiennent le littéral $\neg x_6$, $\neg x_5$ et $\neg x_4$.

3.2 Subsumption dynamique : formulation générale

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

Définition 4 (F-subsumption modulo PU) *Soit $c \in \mathcal{F}$. c est $\mathcal{F}$-subsumée modulo la propagation unitaire ssi $\exists c' \subset c$ tel que $\mathcal{F}|_{\vec{c'}} \models^* \bot$.*

Soient c_1 et c_2 deux clauses de $\mathcal{F}$ telles que c_1 subsumes c_2, alors c_2 est $\mathcal{F}$-subsumed modulo PU.

Comme expliqué auparavant, la subsumption des clauses durant la recherche peut être coûteuse. Dans le cadre proposé, pour réduire le coût calculatoire, on retreint la vérification de la subsumption aux résolvantes intermidiares σ_i et les clauses de la forme $\overrightarrow{cla}(y)$ utilisées pour les générer (clauses codées dans le graphe d'implications).

Définition 5 *Soit $\mathcal{F}$ une formule et $\pi = \langle \sigma_1 \ldots \sigma_k \rangle$ le dérivation de la clause assertive . Pour chaque $\sigma_i \in \pi$, on définit $\mathcal{C}_{\sigma_i} = \{\overrightarrow{cla}(y) \in \mathcal{F} | \exists j \leq i$ tq. $\sigma_j = \eta[y, \overrightarrow{cla}(y), \sigma_{j-1}]\}$ comme l'ensemble de clauses de $\mathcal{F}$ utilisé dans la déduction de σ_i.*

Propriété 1 *Soit $\mathcal{F}$ une formule et $\pi = \langle \sigma_1, \sigma_2, \ldots, \sigma_i, \ldots, \sigma_k \rangle$ le dérivation de la clause assertive . Si σ_i subsume une clause c de $\mathcal{C}_{\sigma_k}$ alors $c \in \mathcal{C}_{\sigma_i}$.*

Preuve 1 *Comme $\sigma_{i+1} = \eta[y, \overrightarrow{cla}(y), \sigma_i]$ tel que $\neg y \in \sigma_i$, nous avons $\sigma_i \not\subset \overrightarrow{cla}(y)$. La prochaine étape de résolution ne peut pas impliquer les clauses contenant le littéral $\neg y$. Dans le cas contraire, le littéral y dans le graphe d'implications peut admettre deux explications possibles (implications), ce qui contredit la définition du graphe d'implications. Par conséquent, σ_i ne peut pas subsumer des clauses de $\mathcal{C}_{\sigma_k} - \mathcal{C}_{\sigma_i}$.*

Propriété 2 *Soit $\mathcal{F}$ une formule et π une dérivation de la clause assertive . Si $\sigma_i \in \pi$ subsume une clause c de $\mathcal{C}_{\sigma_i}$ alors c est $\mathcal{C}_{\sigma_i}$-subsumé modulo PU.*

Preuve 2 *Comme $\sigma_i \in \pi$ est générée à partir de $\mathcal{C}_{\sigma_i}$ par résolution, alors $\mathcal{C}_{\sigma_i} \models \sigma_i$. Par définition de la dérivation de la clause assertive et le graphe d'implications, on a aussi $\mathcal{C}_{\sigma_i} \models^* \sigma_i$ (voir section 2.3). Comme σ_i subsumes c ($\sigma_i \subseteq c$), nous déduisons que $\mathcal{C}_{\sigma_i} \models^* c$.*

La propriété 2 montre que si une clause c codée dans le graphe d'implications est subsumée par σ_i, alors une telle subsumption peut être capturée par la subsumption modulo PU, alors que la propriété 1 mentionne que le test de vérification de la subsumption de σ_i peut être restreint aux clauses de $\mathcal{C}_{\sigma_i}$. Par conséquent, une première approche de subsumption dynamique des clauses peut être définit comme suit : Soit $\pi = \langle \sigma_1, \ldots, \sigma_i, \ldots, \sigma_k \rangle$ une dérivation de la clause assertive . Pour chaque résolvante $\sigma_i \in \pi$, nous appliquons le test de vérification de la subsumption entre σ_i et l'ensemble de clauses $\mathcal{C}_{\sigma_i}$,

Dans la suite, nous montrons que nous pouvons réduire encore plus l'ensemble des clauses dans les tests de subsumption en considérant seulement un sous-ensemble de $\mathcal{C}_{\sigma_i}$. Évidemment, comme σ_i est une résolvante de la dérivation de la clause assertive π, alors il existe deux chemins à partir reliant les nœuds conflictuels x et $\neg x$ respectivement aux nœuds associés aux littéraux de σ_i affectés au niveau du conflit. Par conséquent, on a la propriété suivante.

Propriété 3 *Soit π une dérivation de la clause assertive , $\sigma_i \in \pi$ et $c \in \mathcal{C}_{\sigma_i}$. Si σ_i subsumes c, alors il existe deux chemins liant les nœuds conflictuels x et $\neg x$ respectivement, à un ou plusieurs nœuds associés aux littéraux de la clause σ_i affectés au niveau du conflit.*

La preuve de la propriété est immédiate du moment que $\sigma_i \subset c$. Comme cette propriété est vraie pour σ_i qui est générée par résolution à partir des deux clauses contenant

x et $\neg x$, alors elle reste vraie aussi pour son super-ensemble (c).

Pour une σ_i donnée, la propriété 3 mène à une autre restriction de l'ensemble des clauses à considérer pour les tests de subsumption. Nous considérons que l'ensemble des clauses $\mathcal{P}_{\sigma_i}$, liées (par des chemins) aux nœuds conflictuels x et $\neg x$.

Illustrons cette restriction en utilisant l'exemple 1 (voir aussi la Figure 1). Soit $\pi = \langle \sigma_1, \ldots, \sigma_7 \rangle$ tel que $\sigma_7 = (\neg x_{11} \vee \neg x_{12} \vee \neg x_6 \vee \neg x_1)$. On a $\mathcal{C}_{\sigma_7} = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$ et $\mathcal{P}_{\sigma_7} = \{c_1, c_2, c_4, c_5, c_6, c_8\}$. En effet, à partir du des nœuds de la clause c_3 on a seulement qu'un seul chemin vers le nœud x_9. par conséquent, la clause c_3 peut être enlevée de l'ensemble des clauses à vérifier par subsumption. De façon similaire, partant de la clause c_7 on ne peut atteindre qu'un seul nœud conflictuel en l'occurence x_9 à partir de x_7. Donc, c_7 peut être omise de l'ensemble des clauses susceptibles d'être subsumées.

Propriété 4 *Soit $\pi = \langle \sigma_1, \ldots, \sigma_k \rangle$ une dérivation de la clause assertive . La complexité temporelle de notre approche de subsumption dynamique est en $O(|\mathcal{C}_{\sigma_k}|^2)$.*

Preuve 3 *À partir de la définition de $\mathcal{C}_{\sigma_i}$, nous avons $|\mathcal{C}_{\sigma_i}| = i + 1$. Dans le pire des cas, on doit considérer $i + 1$ tests de subsumption. Donc pour toute σ_i avec $1 \leq i \leq k$, on doit tester $\sum_{1 \leq i \leq k}(i + 1) = \frac{k \times (k+3)}{2}$. Comme $k = |\mathcal{C}_{\sigma_k}|$, la complexité dans le pire des cas est en $O(|\mathcal{C}_{\sigma_k}|^2)$.*

La complexité dans le pire des cas est quadratique si on considère $\mathcal{P}_{\sigma_k} \subset \mathcal{C}_{\sigma_k}$.

3.3 Subsumption dynamique

Dans la section 3.2, nous avons présenté l'approche générale de subsumption dynamique. Sa complexité est quadratique en le nombre de clauses utilisées dans la dérivation de la clause assertive. Comme indiqué dans l'introduction, pour cencevoir une technique de subsumption dynamique et efficace, nous avons besoin de trouver un compromis entre le temps de calcul et la qualité de la simplification. Dans cette section, on propose une restriction du schéma général de subsumption dynamique présenté, appelé subsumption dynamique à la volée. Celle-ci applique la subsumption seulement entre la résolvante courante σ_i et la dernière clause du graphe d'implications utilisée pour sa dérivation. Plus précisemment, supposons que $\sigma_i = \eta[y, c, \sigma_{i-1}]$, nous testons seulement la subsumption entre σ_i et c.

La propriété suivante montre une condition suffistante permettant d'éliminer y de c.

Propriété 5 *Soit π une dérivation de la clause assertive, $\sigma_i \in \pi$ telle que $\sigma_i = \eta[y, c, \sigma_{i-1}]$. Si $\sigma_{i-1} - \{y\} \subseteq c$, alors c est subsumée par σ_i.*

Preuve 4 *Soit $c = (\neg y \vee \alpha)$ et $\sigma_{i-1} = (y \vee \beta)$. Alors $\sigma_i = (\alpha \vee \beta)$. Comme $\sigma_{i-1} - \{y\} \subseteq c$, donc $\beta \subseteq \alpha$. D'où, $\sigma_i = \alpha$ qui subsume $(\neg y \vee \alpha) = c$.*

En considérant les solveurs SAT modernes inculant l'analyse de conflits, l'intégration de l'approche de subsumption dynamique peut être faite avec un coût négligeable. En effet, en utilisant un simple compteur, durant la procédure de l'analyse de conflits, on peut vérifier la condition suffisante donnée dans la propriété 5 en temps constant. En effet, à chaque étape du processus de dérivation de la clause assertive, on génére la prochaine résolvante σ_i à partir de la clause c et l'ancienne résolvante σ_{i-1}. Dans l'implémentation classique de l'analyse de conflit, on peut vérifier en temps constant si un littéral donnée est présent dans la résolvante courante. Par conséquent, durant le parcours de la clause c, on calcule additionnellement le nombre n des littéraux de c qui appartiennent à σ_{i-1}. Si $n \geq |\sigma_{i-1}| - 1$ nous pouvons conclure que c est subsumée par $\sigma_i = \eta[y, c, \sigma_{i-1}]$.

4 Expérimentations

Les expérimentations menées sont effectuées sur un large panel d'instances industrielles et crafted. Toutes les instances sont simplifiées par pré-traitement `SatElite` [8]. Nous avons intégré notre approche de subsumption dynamique dans `Minisat` [9] et `Rsat` [15] et nous avons effectué une comparaison entre les résultats obtenues par les solveurs originaux et ceux incluant l'approche de subsumption dynamique. Tous les tests sont effecutées sur un cluster Xeon 3.2GHz (2 GB RAM). Les Résultats du temps de calcul sont indiqués en secondes.

4.1 Problèmes crafted

Pour ces expérimentations, le temps de calcul limite est fixé à 3 heures. Ces problèmes sont connues pour être difficile pour tous les solveurs DPLL existants. Ils contiennent par exemple les instances "Quasi-group", instances SAT forcées aléatoires, comptage, problèmes sociaux du golfeur, etc.

Le schéma (en échelle logarithmique) donnée dans la partie haute de la Figure 2 détaille les résultats de `Minisat` et `Minisat+SD` sur chaque instance crafted. L'axe x (resp. y) corresponds au temps CPU tx (resp. ty) obtenu par `Minisat` (resp. `Minisat+SD`). chaque point de coordonnées (tx, ty), corresponds à une instance SAT. Les points en-dessous (resp. au-dessus) de la diagonale indiques les instances résolues plus rapidement en utilisant

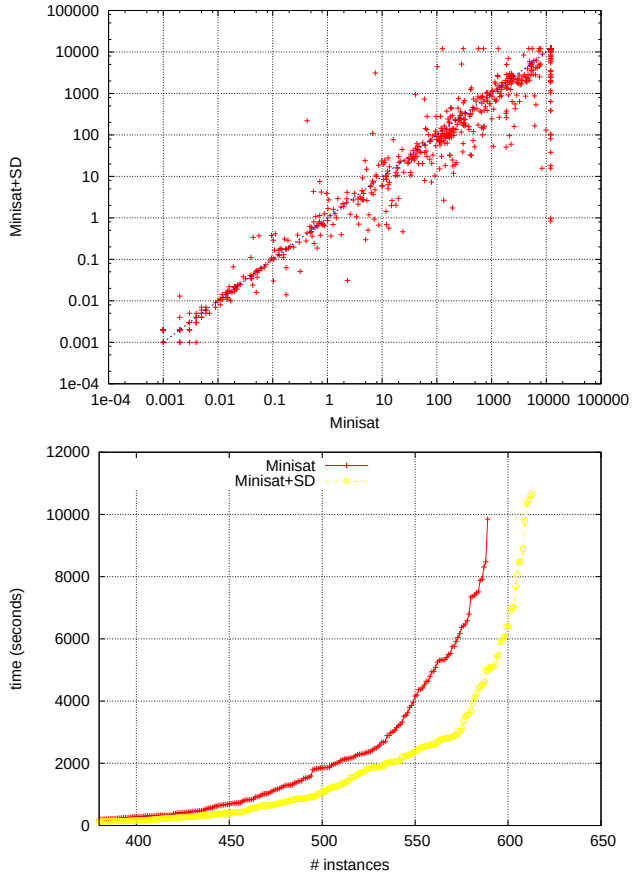

FIG. 2 – Problèmes crafted : `Minisat` vs `Minisat+SD`

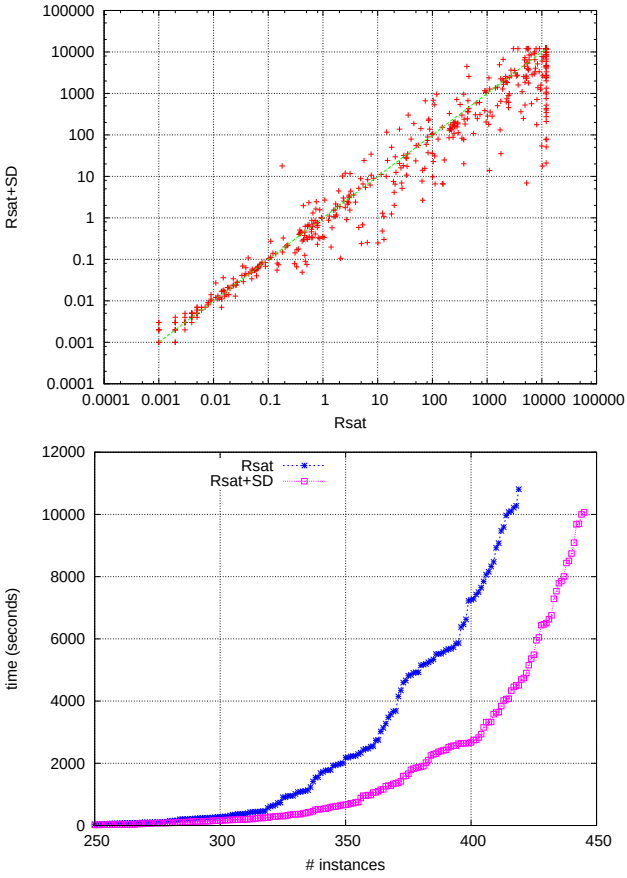

FIG. 3 – Problèmes crafted : `Rsat` vs `Rsat+SD`

la subsumption dynamique i.e., $ty < tx$ (resp. $ty > tx$). Cette figure montre clairement le gain de temps obtenu grâce à l'approche de subsumption dynamique. En nombre de points, nous avons trouvé que 365 instances sont résolues efficacement en intégrant la subsumption dynamique. Dans certains cas, nous avons des gains atteignant un ordre de magnitude. Bien évidemment, il existe des instances où la subsumption dynamque décroît les performances de `Minisat` (178 instances).

La partie en bas de la figure 2 montre les mêmes résultats avec une représentation différente donnant pour chaque technique le nombre d'instances résolues (# instances) en moins de t secondes. Cette figure confirme l'efficacité de l'approche présentée sur ces problèmes. Sur plusieurs classes, le nombre de clauses subsumées est important e.g., `x_*`, `QG_*`, `php_*`, `parity_*`. Sur les `genurq_*`, `mod_*`, et `urquhart_*` les problèmes sont simplifiés chaque conflit sur 4.

La Figure 3 montre les résultats de `Rsat` et `Rsat+SD`. Globalement, on peut voir que l'ajout de la subsumption dynamique à `Rsat` améliore ces performances. Une analyse fine de la partie gauche de la figure 3 montre que `Rsat+SD` résoud plus rapidement 327 instances que `Rsat`, qui lui même résoud 219 problèmes plus efficacement que son son opposé.

À notée que les performances de `Rsat` et `Rsat+SD` sont inférieures à celles de `Minisat` et `Minisat+SD`. Ceci est dû principalement au faite que `Rsat` et `Rsat+SD` utilise une politique de redémarrage plus rapide qui reste moins efficace sur les instances de type crafted.

4.2 Problèmes industriels

Comme pour les instances crafted, sur cette catégorie de problèmes, le temps limite est fixé à 3 heures aussi. La table 1 détaille les résultats sur les problèmes SAT industriels issus des deux compétitions SAT'2007 et SatRace'2008. La première colonne représente les familles d'instances. La seconde (#inst.) indique le nombre total des instances contenu dans chaque famille. Les colonnes suivantes indiquent les résulats respectives pour `Rsat`, `Rsat+SD`, `Minisat` et `Minisat+SD`. Dans chacune de ces colonnes, le premier nombre représente le nombre d'instances résolues et le nombre entre parenthèses indique le nombre d'instances résolues plus rapidement que son opposé. La dernière ligne de la table indique le nombre total de chaque colonne. On peut voir que `Rsat+SD` et `Minisat+SD` sont généralement plus rapides et résolvent plus de problèmes que `Rsat` et `Minisat` respectivement.

La table 2, représente un focus sur quelques familles industrielles. Les améliorations sont relativement importants. Si on considère la famille vmpc, on peut voir que notre simplication dynamique permet d'avoir un gain d'un ordre de magnitude avec `Rsat+SD` (instances 24 et 25). Sur la même famille, sur les 9 instances résolues par `Rsat+SD` et `Rsat`, `Rsat+SD` est meilleure 8 fois. De sont côté `Minisat+SD` est meilleur 5 fois sur 7.

Globalement, nos expérimentations nous ont permis de démontrer deux choses. Premièrement, notre technique ne dégrade pas mais au contraite améliore souvent les performances des solveurs DPLL sur les problèmes indistriels. Second, Elle améliore l'applicabilité de ces algorithmes sur des classes de problèmes qui sont faites pour être difficiles pour eux. Vu que l'implémentation de notre algorithme est aussi simple, nous pensons que de manière globale, elle resprésente une contribution intéréssante dans une recherche de robustesse des solveurs DPLL modernes.

5 Travaux précédents

Dans Darras et al. [5], les auteurs proposent un prétraitement basé sur la propagation unitaire pour la déduction des sous-clauses. En considérant le graphe d'implications généré par le processus de la propagation de contraintes comme un arbre de résolution, l'approche proposée déduit des sous-clauses à partir des clauses originales de la formule. Cependant, l'approche dynamique proposée par les auteurs est coûteuse en temps. L'évaluation expérimentale est donnée seulement en terme de nombre de nœuds.

Dans [17], un algorithme pour maintenir la base de clauses fermée par subsumption est présentée. Elle détecte efficacement et applique la subsumption lorsqu'une clause est ajoutée. En plus, l'algorithme compacte avidement la base des clauses en appliquant récursivement la résolution dans le but de réduire la taille de la base.

"Minimisation de la clause conflit" est introduite dans [14]. Elle est aussi implémentée dans Minisat 1.14 [7] et PicoSAT [4]. Elle élimine les littéraux des clauses apprises en effectuant la résolution de façon récursive avec les clauses du graphe d'implications. Remarquons que dans les expérimentations effectuées les solveurs de base utilisées `Minisat` et `Rsat` implémentent déjà cette technique.

6 Conclusion

Ce travail présente une nouvelle technique de subsumption des formules booléennes sous forme CNF. Il utilise de façon originale l'apprentissage pour réduire les clauses originales ou apprises. À chaque conflit et durant le processus de dérivation de la clause assertive, la subsumption entre les résolvantes intermédiaires générées et des clauses codées dans le graphe d'implications est testée dans le but de les simplifier en vérifiant une condition suffisante. Il est a noter que comme notre technique de subsumption repose sur les clauses utilisées dans la dérivation de la clause assertive, elle tend à simplifier la partie de formule identifiée

familles	# inst.	Rsat	Rsat+SD	Minisat	Minisat+SD
IBM_*	53	15(7)	**17**(**10**)	15(**10**)	15(7)
APro_*	16	12(**7**)	12(5)	**14**(6)	13(**8**)
mizh_*	10	10(**7**)	10(3)	10(5)	10(5)
Partial_*	20	6(2)	**7**(**5**)	1(0)	**2**(**2**)
total_*	20	13(6)	13(**8**)	**10**(5)	9(**6**)
dated_*	20	15(6)	**16**(**10**)	**14**(**10**)	13(4)
braun_*	7	4(1)	4(**3**)	5(2)	5(**3**)
velev_*	10	2 (0)	2(**2**)	**2**(**2**)	1(0)
sort_*	5	2(**2**)	2(0)	2(0)	2(**2**)
manol_*	10	8(3)	8(**5**)	8(**5**)	**9**(4)
vmpc_*	10	9(1)	9(**8**)	6(2)	**7**(**5**)
clause_*	5	3(**2**)	3(1)	3(0)	3(**3**)
cube_*	4	4(2)	4(2)	4(1)	4(**3**)
gold_*	4	2 (2)	2 (**0**)	2(0)	2(**2**)
safe_*	4	2(0)	2(**2**)	1(0)	1(**1**)
simon_*	5	5 (**4**)	5 (1)	5(**3**)	5(2)
block_*	2	2(**2**)	2(0)	2(0)	2(**2**)
dspam_*	10	10(5)	10(5)	10(5)	10(5)
schup_*	3	3(2)	3(1)	3(**2**)	3(1)
post_*	10	8(3)	8(**5**)	5(3)	**6**(3)
ibm_*	20	20(6)	20(**14**)	19(6)	19(**13**)
Total	248	155(70)	**159**(**90**)	141(67)	141(**81**)

TAB. 1 – Problèmes industriels

comme étant importante par la stratégie de recherche basée sur l'activité.

Les résultats expérimentaux montrent que l'intégration de notre méthode au sein de deux solveurs `Minisat` et `Rsat` issus de l'état de l'art de SAT bénéficie particulierement aux problèmes crafted et permet des améliorations significatives sur plusieurs familles industrielles.

Comme travaux futures, nous envisageons d'étendre notre approche en visant une subsumption plus exhaustive des clauses. Une autre voie de recherche consiste à exploiter ce cadre de subsumption pour affiner l'heuristique de choix de variable. En effet, à chaque fois qu'un littéral est éliminé, celà signifie qu'on s'est trompé de clause conflit et que le graphe d'implication doit changer en conséquence et du même coup l'ensemble des variables à pondérer par l'heuristique.

Références

[1] 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.

[2] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 1194–1201. Morgan Kaufmann, 2003.

[3] A Biere and N. Eén. 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)*, 2005.

[4] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 4(1) :75–97, 2008.

[5] S. Darras, G. Dequen, L. Devendeville, B. Mazure, R. Ostrowski, and L. Saïs. Using Boolean constraint propagation for sub-clauses deduction. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming(CP'05)*, pages 757–761, 2005.

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

[7] N. Een and N. Sörensson. Minisat - a sat solver with conflict-clause minimization. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, 2005.

[8] 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.

[9] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International*

instances	Rsat	Rsat+SD	Minisat	Minisat+SD
vmpc_33	5540	**1562**	–	–
vmpc_29	2598	**1302**	–	**1252**
vmpc_30	366	**105**	3111	**2039**
vmpc_27	593	**327**	1159	**637**
vmpc_31	–	–	–	–
vmpc_25	39	**1**	830	**318**
vmpc_26	182	**69**	1239	**1235**
vmpc_34	3366	**944**	–	–
vmpc_24	43	**8**	**82**	210
vmpc_28	**173**	488	**3859**	5478
partial-5-11-s	931	**176**	–	**2498**
partial-5-13-s	503	**71**	3248.38	**669**
partial-5-15-s	**737.064**	825	–	–
partial-10-11-s	1242	**875**	–	–
partial-5-19-s	1134	**498**	–	–
partial-5-17-s	**7437.82**	10610	–	–
partial-10-13-s	–	**3237**	–	–
ibm-2002-04r-k80	90	**33**	**113**	152
ibm-2002-11r1-k45	67	**29**	102	**65**
ibm-2002-18r-k90	265	**157**	1044	**769**
ibm-2002-20r-k75	**36**	185	2112	**668**
ibm-2002-22r-k60	738	**691**	5480	**3434**
ibm-2002-22r-k75	363	**349**	1109	**688**
ibm-2002-22r-k80	**285**	298	894	**642**
ibm-2002-23r-k90	1477	**965**	7127	**2670**
ibm-2002-24r3-k100	273	**256**	**133**	249
ibm-2002-25r-k10	**3104**	3118	**2877**	3172
ibm-2002-29r-k75	353	**248**	**272**	1107
ibm-2002-30r-k85	3853	**592**	–	–
ibm-2002-31_1r3-k30	1203	**652**	1150	**998**
ibm-2004-01-k90	114	**30**	**251**	726
ibm-2004-1_11-k80	394	**222**	559	**329**
ibm-2004-23-k100	**326**	687	3444	**2743**
ibm-2004-23-k80	**465**	563	2060	**1584**
ibm-2004-29-k25	290	**210**	1061	**1017**
ibm-2004-29-k55	533	**16**	558	**124**
ibm-2004-3_02_3-k95	**1**	2	**1**	2

TAB. 2 – Zoom sur quelques familles industrielles

Conference on Theory and Applications of Satisfiability Testing (SAT'03), pages 502–518, 2002.

[10] 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, Madison, Wisconsin, 1998.

[11] 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.

[12] 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, November 1996.

[13] 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.

[14] Alexander Nadel. Backtrack search algorithms for propositional logic satisfiability : Review and innovations. Master's thesis, Master Thesis, the Hebrew University, November 2002.

[15] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of 10th International Conference on Theory and Applications of Satisfiability Testing(SAT)*, pages 294–299, 2007.

[16] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER : Non-increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 276–291, 2004.

[17] Lintao Zhang. On subsumption removal and on-the-fly cnf simplification. In *SAT'2005*, pages 482–489, 2005.