

Une nouvelle méthode hybride pour calculer tous les MSS et tous les MUS

Éric Grégoire Bertrand Mazure Cédric Piette

CRIL-CNRS & IRCICA, Université d'Artois
rue Jean Souvraz SP18, F-62307 Lens Cedex France
{gregoire,mazure,piette}@cril.univ-artois.fr

Résumé

Dans ce papier, nous présentons une nouvelle technique complète permettant le calcul des sous-formules maximales consistantes (MSS) et des formules minimales inconsistantes (MUS) d'un ensemble de clauses booléennes. Cette approche améliore la meilleure technique complète connue de plusieurs manières. Elle utilise à la fois une recherche locale peu coûteuse en ressources et le nouveau concept de clause critique pour améliorer un algorithme complet proposé par Liffiton et Sakallah. Cette hybridation permet l'obtention de gains exponentiels. Ainsi, des résultats expérimentaux montrent que cette nouvelle approche dépasse la meilleure méthode proposée jusque ici.

Mots-clé : Satisfiabilité, Hybridation, MSS, MUS, Recherche locale

1 Introduction

Depuis plus d'une dizaine d'années, SAT, c'est-à-dire le problème consistant à décider si un ensemble de clauses booléennes est satisfiable ou non, a reçu beaucoup d'attention de la part de la communauté des chercheurs en intelligence artificielle. En effet, SAT apparaît comme une pierre angulaire pour de nombreux domaines, tels la logique non-monotone, le raisonnement automatique, la validation de bases de connaissances, la planification ou le diagnostic de pannes. Cependant, la seule connaissance de l'incohérence d'une instance SAT est souvent non satisfaisante, puisque l'on préfère souvent savoir quelle(s) partie(s) de la formule génère(nt) cette inconsistance.

Dans cette optique, le concept de formule minimale inconsistante (ou MUS pour *Minimally Unsatisfiable Subformula*) peut se révéler essentiel, car un MUS représente un ensemble de clauses irréductible pour l'insatisfaisabilité.

En effet, un MUS est un ensemble incohérent de clauses tel que tous ses ensembles propres sont satisfiables. Celui-ci fournit donc une « explication » de l'incohérence qui ne peut être plus petite en terme du nombre de contraintes impliquées. Restaurer la satisfaisabilité d'une instance ne peut se faire sans réparer chacun de ses MUS.

Malheureusement, une même formule peut posséder plusieurs MUS. En fait, dans le pire cas, le nombre de MUS est exponentiel ; une formule SAT composée de n clauses peut en effet exhiber $C_n^{n/2}$ MUS. De plus, le calcul de MUS n'est pas réalisable en pratique dans le pire des cas. En effet, décider si un ensemble de clauses est un MUS est DP-complet [18], et tester si une formule appartient à l'ensemble des MUS (ou clutter) d'une instance incohérente est Σ_2^P -difficile [8]. Fort heureusement, le nombre de MUS s'avère souvent limité dans les problèmes d'applications pratiques. Par exemple, en diagnostic de pannes [13], on suppose le plus souvent qu'une seule erreur est présente, ce qui induit un nombre limité de MUS.

Le concept dual à cette notion de MUS est celle de *sous-formule maximale satisfiable* (MSS) d'une formule SAT, le complémentaire d'un MSS étant appelé CoMSS. Les ensembles complets de MUS et de MSS s'encodent implicitement l'un l'autre [15]. Plus précisément, l'ensemble des CoMSS est un ensemble intersectant de l'ensemble des MUS et représente les ensembles minimaux de clauses qui peuvent être supprimées dans le but de restaurer la consistance de la formule. Dans ce papier, nous nous intéressons aux approches exhaustives calculant ces trois ensembles corrélés dans le cadre clausal booléen.

Récemment, plusieurs algorithmes ont été proposés dans le but de faire l'approximation, voire le calcul exact d'un MUS ou d'un MSS, à la fois dans le cadre booléen et pour d'autres types de contraintes. Certains travaux concernent des classes restreintes de formules ou ne sont applicables en pratique que pour de très petites instances. Parmi ces

approches, on trouve la méthode proposée dans [5], qui montre comment un MUS peut être extrait en temps polynomial à travers des techniques de programmation linéaire, pour les formules vérifiant une propriété dite « integral property ». Toutefois, seulement quelques classes de formules obéissent à une telle propriété (principalement les formules Horn, Horn-renomables, Horn étendues et bien équilibrées). Mentionnons également [3][6][10], qui sont d’autres travaux importants sur la complexité et les aspects algorithmiques de l’extraction de MUS pour des classes spécifiques de formules. Dans [4], une méthode est proposée dans le but d’approcher un MUS grâce à une recherche adaptative guidée par la « difficulté » supposée des clauses. Une technique décrite dans [21] permet le calcul d’un MUS par l’apprentissage de *nogoods* impliqués dans la dérivation par résolution de la clause vide. Dans [16], une technique complète qui calcule cette fois le plus petit des MUS d’une formule est introduite. On trouve également dans la littérature une approche basée sur DPLL [17] qui consiste à marquer les clauses pour approximer un MUS. Enfin, dans [11], Grégoire, Mazure et Piette ont proposé une technique heuristique pour approcher voire calculer un MUS, qui surpasse en pratique les meilleures méthodes connues. De surcroît, ces mêmes auteurs ont introduit le concept de *couverture inconsistante* [12] (ou « inconsistent cover ») pour gérer en pratique le nombre potentiellement exponentiel de MUS dans une formule. L’intérêt principal de la notion de couverture inconsistante est de représenter un ensemble de MUS qui couvre suffisamment de causes d’incohérence pour pouvoir restaurer la satisfaisabilité de l’instance si chacune de ces causes est « réparée ». Bien qu’une couverture inconsistante ne couvre pas l’ensemble des MUS présents dans une formule, elle fournit cependant une série suffisante d’explications minimales d’incohérence pour expliquer et potentiellement réparer suffisamment de cause(s) indépendante(s) de l’incohérence pour restaurer la satisfaisabilité de l’ensemble de la formule.

Toutes ces méthodes sont incomplètes dans le sens où elles ne retournent pas nécessairement tous les MUS d’une formule. Toutefois, dans certains domaines d’applications, il peut s’avérer nécessaire de calculer l’ensemble de *tous* les MUS, parce que le diagnostic d’infaisabilité est difficile, sinon impossible, sans une connaissance complète de toutes ses causes [15]. Évidemment, une telle technique peut se révéler inapplicable en pratique quand le nombre de MUS lui-même ne permet pas de rendre envisageable le fait de simplement les lister un à un. De même, le nombre de MSS (et donc de CoMSS) peut être exponentiel dans le pire des cas. Notons que dans de nombreux domaines de l’intelligence artificielle tels que la révision de croyances (cf. [2]) induisent des notions conceptuelles de la gestion de l’insatisfaisabilité qui requièrent le calcul des ensembles complets de MUS, MSS, et CoMSS dans le pire cas, même lorsque des concepts épistémologiques supplé-

mentaires comme la stratification sont ajoutés au cadre logique.

Dans cet article, l’attention est portée sur les méthodes complètes. Nous introduisons une nouvelle technique algorithmique pour calculer l’ensemble des MUS, MSS et CoMSS d’une instance SAT, qui possède clairement des limitations en pratique. Elle améliore la meilleure méthode complète connue, à savoir l’approche de Liffiton et Sakallah [15] (notée L&S), qui avait été à son tour prouvée plus efficace que les anciennes méthodes de Bailey et Stuckey [1], et de de la Banda, Stuckey et Wazny [7], qui avaient été introduites dans différents contextes.

Notre approche possède deux caractéristiques principales. D’abord, il s’agit d’une hybridation entre l’approche complète L&S et un prétraitement à base de recherche locale. Cette dernière est en effet utilisée comme un oracle pour trouver des CoMSS potentiels d’une instance SAT, qui sont eux-même un ensemble intersectant de l’ensemble des MUS. Nous montrons qu’une telle hybridation peut conduire à des gains d’ordre exponentiel. Par ailleurs, l’efficacité de l’approche est liée au concept de clause critique, qui s’est déjà révélé un ingrédient efficace dans notre approche effectuant le calcul d’un MUS [11].

La suite de ce papier est organisée comme suit. Tout d’abord, les notions élémentaires autour du problème SAT, ainsi que celles concernant les MUS et les concepts duaux de MSS et CoMSS sont présentées. Ensuite, l’approche complète de Liffiton et Sakallah est succinctement décrite. Dans la section 4, nous montrons comment cette technique peut être efficacement couplée avec une recherche locale préliminaire, en utilisant le concept de clause critique. Il est également montré pourquoi ce prétraitement est théoriquement viable, d’un point de vue calculatoire. Enfin, avant de conclure, nous comparons cette nouvelle approche avec celle de Liffiton et Sakallah sur un ensemble varié de benchmarks.

2 Notions préliminaires

Dans cette section, nous fournissons les notions basiques à propos de SAT, MUS, MSS et CoMSS.

Soit \mathcal{L} le langage logique standard construit sur un ensemble fini de variables booléennes, notées a, b, c , etc. Les symboles \wedge, \vee, \neg et \Rightarrow représentent les connecteurs conjonctifs, disjonctifs, négatifs et implicatifs standard, respectivement. Les formules et les clauses sont notées avec des lettres majuscules telles que C . Les ensembles de formules sont quant à elles représentées par des lettres grecques comme Γ ou Σ . Une interprétation est une fonction qui assigne un élément de l’ensemble $\{\text{vrai}, \text{faux}\}$ à chaque variable booléenne. Une formule est satisfiable s’il existe au moins une interprétation (appelée alors modèle) qui la satisfait, c’est-à-dire lui assignant la valeur de vérité *vrai*. Les interprétations sont notées par des lettres ma-

jusques telles que I et sont représentées par l'ensemble des littéraux qu'elles satisfont. De plus, toute formule de \mathcal{L} peut être représentée (en préservant sa satisfaisabilité) en utilisant un ensemble (interprété comme une conjonction) de clauses, où une clause est une disjonction finie de littéraux, et un littéral une variable booléenne ou sa négation. SAT est le problème NP-complet qui consiste à vérifier si un ensemble de clauses booléennes est satisfiable, c'est-à-dire s'il existe une interprétation qui satisfasse toutes ses clauses ou non.

Quand une instance SAT est insatisfiable (ou incohérente), elle exhibe au moins une formule minimale inconsistante (ou MUS pour *Minimally Unsatisfiable Subformula*).

Définition 1 *Un MUS Γ d'une instance SAT Σ est un ensemble de clauses tel que*

1. $\Gamma \subseteq \Sigma$;
2. Γ est insatisfiable ;
3. $\forall \Delta \subset \Gamma$, Δ est satisfiable.

Exemple 1 *Soit $\Sigma = \{a, \neg c, \neg b \vee \neg a, b, \neg b \vee c\}$. Σ possède deux MUS : $\{a, b, \neg b \vee \neg a\}$ et $\{\neg c, b, \neg b \vee c\}$.*

Le concept dual à la notion de MUS est celui de *sous-formule maximale satisfiable* (*Maximal Satisfiable Subset* ou MSS) d'une instance SAT.

Définition 2 *Un MSS Γ d'une instance SAT Σ est un ensemble de clauses tel que :*

1. $\Gamma \subseteq \Sigma$;
2. Γ est satisfiable ;
3. $\forall \Delta \subseteq (\Sigma \setminus \Gamma)$ tel que $\Delta \neq \emptyset$, $\Gamma \cup \Delta$ est insatisfiable.

L'ensemble complémentaire d'un MSS par rapport à une instance SAT est appelé *CoMSS*.

Définition 3 *Le CoMSS d'un MSS Γ d'une instance SAT Σ est donné par $\Sigma \setminus \Gamma$.*

Exemple 2 *Considérons la formule Σ de l'exemple précédent. Σ exhibe cinq CoMSS : $\{b\}$, $\{\neg c, a\}$, $\{\neg c, \neg b \vee \neg a\}$, $\{\neg b \vee c, \neg b \vee \neg a\}$ et $\{\neg b \vee c, a\}$.*

Comme démontré dans [15], ces concepts sont corrélés. Un CoMSS contient en effet au moins une clause de chaque MUS. Plus précisément, un CoMSS est un ensemble intersectant irréductible de l'ensemble des MUS. D'une manière duale, chaque MUS d'une instance SAT est également un ensemble intersectant irréductible des CoMSS. De cette façon, comme souligné dans [15], bien que le MINIMAL-HITTING-SET soit un problème NP-difficile, l'irréductibilité est moins gourmande en ressources que la cardinalité minimale. En fait, un MUS peut même être généré en temps polynomial à partir de l'ensemble des CoMSS.

3 L'approche exhaustive de Liffiton et Sakallah

L'approche de Liffiton et Sakallah [15] (notée L&S) pour calculer l'ensemble des MUS est basée sur cette dualité forte entre les MUS et les MSS. Elle est à notre connaissance la méthode actuelle la plus efficace. Son principe est de calculer préalablement l'ensemble des MSS avant d'en déduire l'ensemble correspondant de MUS. Dans cet article, nous nous concentrons sur la première étape de L&S ; nous proposons en effet une amélioration sur cette partie de l'algorithme et allons adopter la seconde étape telle quelle.

L&S est intégré dans un solveur SAT moderne, ce qui permet l'utilisation des récents progrès algorithmiques sur SAT (structures « paresseuses », prétraitement de la formule, apprentissage de « nogoods », etc.). Grossièrement, chaque clause $C_i = x_1 \vee \dots \vee x_m$ d'une instance SAT est augmentée par la négation d'une nouvelle variable y_i , dite « sélectrice », pour devenir $C'_i = x_1 \vee \dots \vee x_m \vee \neg y_i$. En résolvant cette nouvelle formule, l'assignation de y_i à *faux* a pour effet de désactiver, ou supprimer C_i de l'instance. Un MSS peut donc être obtenu en exhibant une interprétation qui assigne le plus petit nombre possible de variables y_i à *faux*. L'algorithme utilise une méthode permettant un calcul incrémental de cet ensemble de MSS. Une borne sur le nombre de variables y_i qui sont autorisées à être falsifiées est fixée. Pour chaque valeur de cette borne, initialisée à 0 et incrémentée de 1 à chaque itération, une recherche exhaustive est effectuée pour déterminer toutes les interprétations satisfaisant la formule augmentée C'_i , et ainsi calculer les CoMSS dont la taille est égale à cette borne. Quand une solution est trouvée, elle est enregistrée, et une clause correspondante empêchant de recalculer cette solution (ainsi que tous ses sur-ensembles) est insérée. Cette « clause bloquante » est constituée de la disjonction des variables y_i des clauses formant le CoMSS trouvé.

Avant d'effectuer une nouvelle recherche avec la borne suivante, l'algorithme vérifie que la nouvelle instance augmentée des clauses bloquantes est toujours satisfaisable sans aucune borne sur les variables y_i . Si ce n'est pas le cas, l'ensemble entier de CoMSS a été extrait, et l'algorithme termine son exécution.

La seconde partie de l'algorithme calcule l'ensemble complet des MUS à partir des CoMSS trouvés de manière directe. L'approche que nous présentons considère cette étape telle quelle.

4 Recherche locale et clauses critiques

Dans cette section, nous montrons comment la méthode présentée précédemment peut être grandement améliorée en l'hybridant avec une recherche locale préliminaire, qui jouera le rôle d'oracle pour le processus de recherche exhaustive présenté précédemment. Cette nouvelle approche

Algorithme 1 : Approximation par Recherche Locale

Entrées : Une formule CNF Σ
Sorties : Un ensemble de CoMSS candidats

```
1 début
2    $cand \leftarrow \emptyset$ ;
3    $\#echec \leftarrow 0$ ;
4    $I \leftarrow \text{genere\_interpretation\_aleatoire}()$ ;
5   tant que ( $\#echec < \#NB\_ECHECS\_AUTORISES$ ) faire
6      $nouveauCand \leftarrow \text{FAUX}$ ;
7     pour  $j = 1$  to  $\#FLIPS$  faire
8       Soit  $\Delta$  l'ens. des clauses falsifiées par  $I$ ;
9       si ( $\forall C \in \Delta, C$  est critique)
10      et  $\Delta$  n'est pas impliqué par  $cand$  alors
11         $\text{supprimeEnsembleImpliqué}(\Delta, cand)$ ;
12         $cand \leftarrow \Delta \cup cand$ ;
13         $nouveauCand \leftarrow \text{VRAI}$ ;
14       $\text{flip}(I)$ ;
15    si non( $nouveauCand$ ) alors
16       $\#echec \leftarrow \#echec + 1$ ;
17  retourner  $cand$ ;
18 fin
```

est baptisée Hycam (pour HYbridization for Computing All Muses).

En premier lieu, présentons notre méthode de manière intuitive. Clairement, une (rapide) recherche locale initiale, exécutée à la recherche d'un modèle, peut rencontrer certains MSS (par rapport à l'ensemble d'interprétations parcourues). Quand ce phénomène se produit, il peut être utile de conserver les CoMSS correspondants, pour éviter d'avoir à les calculer par la suite de manière complète. De plus, plutôt que de vérifier si nous sommes en présence d'un véritable MSS ou non, on préfère enregistrer le CoMSS *candidat* correspondant; celui-ci sera vérifié durant la partie complète de l'algorithme.

Évidemment, nous avons à étudier quelles interprétations parcourues par la recherche locale peuvent conduire à de bons candidats aux CoMSS et des critères doivent être définis pour conserver un nombre limité de CoMSS potentiels. Dans cette optique, le concept de clause critique peut se révéler extrêmement utile, dans le sens où il fournit une condition nécessaire à la « CoMSS-candidature » qui peut être vérifiée très rapidement. Quand un ensemble de candidats aux CoMSS est enregistré, l'approche incrémentale de Liffiton et Sakallah peut alors exploiter cette information d'une manière efficace.

D'une manière plus détaillée, un algorithme de recherche locale est exécuté sur l'instance SAT initiale. Le but est d'enregistrer autant de CoMSS potentiels que possible, en se basant sur le fait que cette famille d'algorithmes convergent vers les minima locaux, ce qui peut se trans-

Algorithme 2 : Hycam

Entrées : Une formule CNF Σ
Sorties : L'ensemble des CoMSS de Σ

```
1 début
2    $cand \leftarrow \text{RL\_approximation}(\Sigma)$ ;
3    $k \leftarrow 0$ ;
4    $\Sigma_y \leftarrow \text{AjoutSelecteurClauses\_Yi}(\Sigma)$ ;
5    $MSS \leftarrow \emptyset$ ;
6   tant que  $\text{SAT}(\Sigma_y)$  faire
7      $\text{supprimeEnsembleImpliqué}(\{\Sigma \setminus C \mid C \in$ 
8        $MSS\}, cand)$ ;
9      $\Sigma_y \leftarrow \text{ClausesBloquantesTaille}(k, cand)$ ;
10     $MSS \leftarrow MSS \cup \{\Sigma \setminus C \mid C \in cand \text{ et}$ 
11       $|C| = k\}$ ;
12     $MSS \leftarrow MSS \cup \text{SAT\_avec\_borne}(k, \Sigma_y)$ ;
13     $k \leftarrow k + 1$ ;
14  retourner  $MSS$ ;
15 fin
```

crire en de bonnes approximations de MSS. Une approche directe consiste à conserver pour chaque interprétation parcourue l'ensemble de clauses falsifiées. Cependant, de manière évidente, il est inutile d'enregistrer les sur-ensembles de CoMSS candidats; ceux-ci ne sont clairement pas de véritables CoMSS, puisqu'ils ne sont pas minimaux par rapport à l'inclusion ensembliste. Dans ce but, nous avons adapté la technique proposée par Zhang dans [20] aux ensembles de clauses pour enregistrer les CoMSS candidats minimaux parmi l'ensemble de clauses falsifiées déjà calculé. De plus, nous avons également exploité le concept de clause critique, qui s'est révélé efficace dans la localisation de MUS et de couverture inconsistante, quand il est allié à un algorithme incomplet de recherche locale [11][12].

Définition 4 Une clause C est unisatisfaite par une interprétation I si et seulement si exactement un littéral de C est satisfait par I . Une clause C falsifiée par l'interprétation I est critique par rapport à I si et seulement si l'opposé de chaque littéral de C appartient à au moins une clause unisatisfaite par I .

Intuitivement, une clause critique est donc une clause falsifiée qui nécessite qu'une autre clause soit à son tour falsifiée pour qu'elle soit satisfaite, en effectuant un unique *flip*. La propriété suivante montre comment ce concept permet d'éliminer facilement de mauvais candidats CoMSS.

Propriété 1 Soient Σ une instance SAT et I une interprétation. Soit Γ un ensemble non-vide de Σ tel que toutes les clauses de Γ soient falsifiées par I . Quand au moins une clause de Γ n'est pas critique par rapport à I , alors Γ n'est pas un CoMSS de Σ .

Preuve Par définition, quand une clause C_f de Γ n'est pas critique par rapport à I , il existe un littéral $c \in C_f$ dont la valeur de vérité peut être inversée (ou « *flippée* ») sans falsifier aucune autre clause de Σ . Γ n'est donc pas minimal et ne peut être un CoMSS de Σ . \square

En pratique, tester si les clauses falsifiées sont critiques peut s'effectuer efficacement et évite à avoir à enregistrer de trop nombreux CoMSS candidats, ce qui peut s'avérer assez lourd en conjonction avec la gestion de ces ensembles de clauses par [20].

En utilisant ces caractéristiques, une recherche locale est exécutée sur l'instance SAT initiale, et retourne une série de CoMSS candidats. Cette information se montre efficace et permet d'améliorer en pratique l'étape complète de L&S.

L'algorithme L&S est incrémental dans le sens où il calcule progressivement des CoMSS de plus en plus grands. Après que n itérations aient été effectuées, tous les CoMSS de cardinalité n ont été calculés. Clairement, si on a enregistré des CoMSS candidats contenant exactement $n + 1$ clauses qui ne sont pas des sur-ensembles des CoMSS calculés, nous avons la garantie que ceux-ci sont de vrais CoMSS. Ainsi, nous n'avons pas à les calculer, et les clauses bloquantes correspondantes peuvent être insérées directement. De plus, il n'est pas nécessaire d'effectuer le test SAT au terme de la $n^{\text{ème}}$ itération, puisque nous avons connaissance de l'existence de plus grands CoMSS.

Il est assez aisé de comprendre que l'insertion de clauses bloquantes permet d'éviter à la fois des tests NP-complets et CoNP-complets. Ceci est illustré par l'exemple suivant.

Exemple 3 Soient Σ une instance insatisfiable, et Σ' l'instance SAT correspondante augmentée des variables y_i , sélecteurs de clauses de L&S.

$$\Sigma = \begin{cases} C_0 : (d) & C_1 : (b \vee c) \\ C_2 : (a \vee b) & C_3 : (a \vee \neg c) \\ C_4 : (\neg b \vee \neg e) & C_5 : (\neg a \vee \neg b) \\ C_6 : (a \vee e) & C_7 : (\neg a \vee \neg e) \\ C_8 : (b \vee e) & C_9 : (\neg a \vee b \vee \neg c) \\ C_{10} : (\neg a \vee b \vee \neg d) & C_{11} : (a \vee \neg b \vee c) \\ C_{12} : (a \vee \neg b \vee \neg d) & \end{cases}$$

Σ est une formule insatisfiable composée de 13 clauses qui sont construites sur 5 variables. Σ contient 3 MUS, représentés par la Figure 1, et admet 19 MSS. Supposons que L&S et HYCAM soient tous deux lancés avec cette instance en entrée. Ses clauses sont augmentées par les littéraux négatifs $\neg y_i$, jouant le rôle de sélecteurs de clauses. Supposons également que la recherche locale effectuée par HYCAM fournisse 4 CoMSS candidats : $\{C_5\}$, $\{C_3, C_2\}$, $\{C_0, C_1, C_2\}$ et $\{C_3, C_8, C_{10}\}$.

Si les variables à affecter sont choisies dans l'ordre lexicographique, alors a et b sont d'abord assignées à *vrai* et C_5 est falsifiée. L&S tente alors de prouver que cette

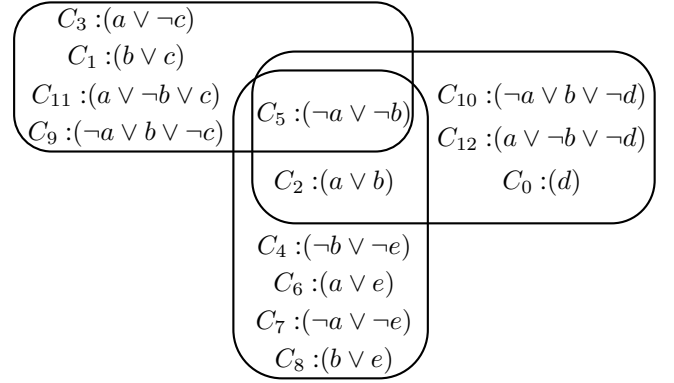


FIG. 1: MUS de l'exemple 3

clause forme un CoMSS, ce qui nécessite un test NP-complet (il a en effet à trouver un modèle à la formule $\Sigma \setminus \{\neg a \vee \neg b\} \cup \{a, b\}$). Au contraire, avec HYCAM, la clause bloquante y_5 est ajoutée avant la première itération de l'algorithme complet, puisque la recherche locale a auparavant retourné ce CoMSS. En conséquence, quand a et b sont affectées à *vrai*, l'algorithme DPLL fait immédiatement un retour en arrière (« *backtrack* »), puisque l'ensemble insatisfiable $\{y_5, \neg y_5\}$ a été obtenu, sans nécessiter aucun test NP-complet.

D'une manière similaire, l'introduction de clauses sélectrices additionnelles par HYCAM réduit le nombre de tests CoNP-complets effectués. Par exemple, supposons que e soit la première variable assignée (à *faux*) et que les variables suivantes soient affectées dans l'ordre lexicographique. Quand a et b sont affectées à *vrai*, L&S essaie de nouveau de prouver que $\{C_5\}$ est un CoMSS. Puisque $\neg e$ est une conséquence logique de $\Sigma \setminus \{\neg a \vee \neg b\} \cup \{a, b\}$, il n'existe aucun modèle à $\Sigma \setminus \{\neg a \vee \neg b\} \cup \{a, b, \neg e\}$. Clairement, un tel test est dans CoNP. Grâce aux CoMSS candidats fournis préalablement par HYCAM, on peut éviter de parcourir cette partie de l'espace de recherche. En effet, puisque l'on sait que $\{C_5\}$ est un CoMSS de Σ , quand a et b sont affectées à *vrai*, plus aucun test CoNP n'est nécessaire, avec cette interprétation partielle. Ceci montre clairement comment une recherche locale, utilisée comme oracle et peu coûteuse en ressources, permet d'éviter de nombreux tests NP et CoNP.

5 Evaluation expérimentale

HYCAM a été implémenté et comparé à L&S d'un point de vue pratique. Pour ces deux algorithmes, l'étape complète est basée sur l'architecture de MiniSat [9], qui est à ce jour l'un des meilleurs solveurs SAT. Comme cas d'étude, Walksat [14] a été utilisée pour le prétraitement incomplet. Les nombres de *tries* et de *flips* effectués par la recherche locale ne sont pas fixes, mais dépendent de l'efficacité de cette méthode à trouver de nouveaux CoMSS.

Instance	(#v,#c)	#MSS	#CoMSS		L&S (sec.)	HYCAM (sec.)
			cand.	réels		
hole6	(42,133)	133	133	133	0.040	0.051
hole7	(56,204)	204	204	204	0.75	0.33
hole8	(72,297)	297	293	293	33	1.60
hole9	(90,415)	415	415	415	866	30
hole10	(110,561)	561	559	559	7159	255
x1_16	(46,122)	122	122	122	0.042	0.041
x1_24	(70,186)	186	186	186	7.7	0.82
x1_32	(94,250)	250	241	241	195	28
x1_40	(118,314)	314	314	314	2722	486

TAB. 1: L&S vs HYCAM sur des instances globalement insatisfiables

En fait, à chaque « étape », on effectue un petit nombre de *flips*, et si aucun nouveau candidat n'est capturé, un compteur d'échecs est incrémenté. Quand ce compteur atteint un seuil (expérimentalement fixé à 30), on considère qu'aucun nouveau candidat ne peut être obtenu « facilement » (i.e. sans effectuer un grand nombre de mouvements dans l'espace de recherche) par cette méthode. Cette technique de terminaison offre un bon compromis entre le nombre de CoMSS extraits et le temps passé à leur découverte. D'ailleurs, pour toutes les expérimentations effectuées, cette étape préliminaire ne dépasse pas 5% du temps de calcul global de HYCAM.

Nos tests expérimentaux ont été conduits sur des processeurs Intel Xeon 3GHz sous Linux CentOS 4.1. (kernel 2.6.9) avec 2 Go de mémoire vive. Dans l'ensemble des tables reportant les expérimentations menées, sont notés de gauche à droite : le nom de l'instance testée (*Instance*), ses nombres de variables et clauses (*#v,#c*), le nombre de MSS extraits (*#MSS*), les nombres de CoMSS candidats et réels calculés par la recherche locale de HYCAM (*#CoMSS cand.*, *réel*, respectivement) et enfin le temps en secondes mis par L&S et HYCAM pour effectuer le calcul. La Table 3 contient de plus les colonnes « #MUS » indiquant le nombre de MUS de l'instance et « MSS→MUS » le temps nécessaire à leur calcul. Pour toutes ces expérimentations, une limite de temps de 3 heures CPU a été respectée. Au delà de cette limite, la note *time out* est reportée.

Tout d'abord, dans la Table 1, nous présentons des résultats expérimentaux sur le calcul de MSS sur le célèbre problème des pigeon ainsi que sur les benchmarks « *xor-chains* » [19], qui sont des instances globalement inconsistantes, dans le sens où retirer n'importe laquelle de leurs clauses restaure la consistance de la formule. Ce sont donc de véritables MUS. Clairement, de telles instances possèdent un nombre de CoMSS égal au nombre de clauses dont elles sont composées, et chacun est de taille 1. On a donc la garantie d'un nombre polynomial (et même linéaire) de CoMSS. En pratique, un écart significatif peut être observé en faveur de HYCAM. De surcroît, la différence de temps grandit avec la taille de l'instance. En fait, pour ces instances, la recherche locale réussit la plupart du

temps à calculer l'ensemble des CoMSS, et l'étape complète est souvent réduite à un test d'insatisfaisabilité. Au contraire, L&S doit explorer de nombreux nœuds supplémentaires dans l'espace de recherche pour retourner ces CoMSS.

Dans la Table 2, les résultats expérimentaux sur des benchmarks plus difficiles, extraits de la compétition SAT annuelle [19], sont reportés. Le nombre de leur CoMSS est souvent exponentiel, et leur calcul exhaustif est donc impossible en pratique. Nous avons donc réduit la recherche aux CoMSS de taille inférieure à 5 clauses. Comme ces résultats le montrent, HYCAM surpasse L&S. Par exemple, considérons l'instance *rand_net40-30-10*. Cette formule contient 5831 MSS (dont le CoMSS a une taille inférieure à 5). L&S et HYCAM fournissent ce résultat en 1748 et 386 secondes, respectivement. Pour l'instance *ca256*, HYCAM extrait 9882 MSS en moins de 5 minutes tandis que L&S ne peut les calculer en 3 heures de temps CPU. Notons également que HYCAM peut également retourner des CoMSS contenant 5 clauses après que le calcul soit effectué, puisque tous les ensembles de cette taille qui ne sont pas des sur-ensembles de CoMSS calculés sont également de véritables CoMSS.

Dans la Table 3, des résultats sur des benchmarks difficiles pour le calcul de l'ensemble des MSS et des MUS sont donnés. Comme expliqué plus haut, les deux approches, que ce soit L&S ou HYCAM, nécessitent l'obtention préalable de tous les MSS pour calculer l'ensemble des MUS. Comme le calcul des MSS est permis en un moindre coût par HYCAM, celui-ci permet d'avoir l'ensemble complet des MUS pour de nombreuses formules, et ceci plus rapidement qu'avec L&S. Clairement, quand le nombre de MSS ou de MUS est exponentiel, les calculer et les énumérer exhaustivement se révèle impossible en pratique.

Par exemple, L&S n'a pas été capable de calculer tous les MSS de l'instance *php-012-011* en 3 heures de temps CPU, et ne peut donc découvrir son unique MUS. HYCAM l'extrait quant à lui en 2597 secondes. Sur tous les benchmarks ayant un seul MUS, ou un nombre non-exponentiel, HYCAM est clairement plus efficace que L&S. Ainsi, L&S et HYCAM découvrent les 32 MUS de

Instance	(#v,#c)	#MSS	#CoMSS		L&S	HYCAM
			cand.	réels	(sec.)	(sec.)
rand_net40-25-10	(2000,5921)	5123	4318	2729	893	197
rand_net40-25-5	(2000,5921)	4841	6950	598	650	174
rand_net40-30-10	(2400,7121)	5831	3458	2405	1748	386
rand_net40-30-1	(2400,7121)	7291	4380	662	1590	1325
rand_net40-30-5	(2400,7121)	5673	2611	2507	2145	402
ca032	(558,1606)	1173	1159	1159	4	1
ca064	(1132,3264)	2412	2324	2263	59	3
ca128	(2282,6586)	4899	2878	2422	691	18
ca256	(4584,13236)	9882	9553	9064	<i>time out</i>	277
2pipe	(892,6695)	3571	2094	1849	130	36
2pipe_1_ooo	(834,7026)	3679	1822	1587	52	30
2pipe_2_ooo	(925,8213)	5073	2286	1825	148	61
3pipe_1_ooo	(2223,26561)	17359	7327	3481	5153	2487
am_5_5	(1076,3677)	1959	3250	65	68	57
c432	(389,1115)	1023	1019	1019	4	1
c880	(957,2590)	2408	2141	1866	28	3
bf0432-007	(1040,3668)	10958	3332	2136	233	98
velev-sss-1.0-cl	(1453,12531)	4398	2987	2154	1205	513

TAB. 2: L&S vs HYCAM sur différentes instances SAT difficiles

d1x2_aa en 3,12 et 0,94 secondes, respectivement. On remarque en outre que le temps additionnel passé à calculer tous les MUS à partir des MSS est souvent très court quand leur nombre rend ce problème traitable.

6 Conclusions et recherches futures

Calculer tous les MSS, CoMSS et MUS est un problème hautement complexe, et pose des difficultés insolubles en pratique dans le pire des cas. Cependant, ce problème peut prendre du sens quand il s'agit de les calculer sur des formules issues de problèmes réels. Dans ce papier, nous avons amélioré de plusieurs façons la technique complète la plus efficace à ce jour, c'est-à-dire la méthode proposée par Liffiton et Sakallah. Nos résultats expérimentaux montrent de grands gains en pratique dans l'extraction des MSS, des CoMSS et des MUS. Une de ces caractéristiques les plus intéressantes est son caractère *anytime* pour le calcul des MSS. Des MSS de taille croissante sont en effet calculés successivement. Grâce à cela, on peut imposer une borne maximale sur la taille des CoMSS extraits, limitant les ressources nécessaires à leur extraction. L&S et HYCAM se montrent plus adaptés à extraire les ensembles complets de MSS et CoMSS que ceux de MUS. En effet, cette procédure implique le calcul préliminaire des MSS (et donc des CoMSS). Ainsi, nous nous accordons avec Liffiton et Sakallah à penser qu'une voie intéressante pour de futures recherches concerne la façon selon laquelle les MUS pourraient être calculés progressivement à partir de l'ensemble grandissant de MSS.

De nombreuses recherches en intelligence artificielle ont étudié la manipulation de MUS, MSS ou CoMSS, comme le diagnostic de pannes, la révision de croyances, la gestion de l'inconsistance sur des bases de connaissances ou de croyances, etc. Ces études sont souvent conduites d'un point de vue conceptuel ou dans l'étude de la complexité, exclusivement. Nous pensons que les progrès algorithmiques tels que ceux décrits dans cet article peuvent s'avérer utiles pour traiter ces problèmes en pratique. Nous projetons donc de nous concentrer sur la dérivation d'algorithmes spécifiques pour ces différents problèmes d'intelligence artificielle, en exploitant des résultats comme ceux décrits dans ce papier.

Références

- [1] J. Bailey and P. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL'05)*, pages 174–186, 2005.
- [2] B. Bessant, É Grégoire, P. Marquis, and L. Saïs. *Iterated Syntax-Based Revision in a Nonmonotonic Setting*, volume 22 of *Applied Logic*, chapter of Frontiers in Belief Revision, pages 369–391. Kluwer Academic Publishers, Applied Logic Series, 2001.
- [3] H.K. Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107(1–3):83–98, 2000.

Instance	(#v,#c)	#MSS	#CoMSS		L&S (sec.)	HYCAM (sec.)	#MUS	MSS→MUS (sec.)
			cand.	réels				
mod2-3cage-unsat-9-8	(87, 232)	232	232	232	3745	969	1	0.006
mod2-rand3bip-unsat-105-3	(105, 280)	280	280	280	2113	454	1	0.008
2pipe	(892, 6695)	10221	3142	1925	298	226	> 211 000	time out
php-012-011	(132, 738)	738	734	734	time out	2597	1	0.024
hcb3	(45, 288)	288	288	288	10645	6059	1	0.006
1dlx_c_mc_ex_bp_f	(776, 3725)	1763	946	665	10.4	6.8	> 350 000	time out
hwb-n20-02	(134, 630)	622	588	583	951	462	1	0.01
hwb-n22-02	(144, 688)	680	627	626	2183	811	1	0.025
ssa2670-141	(986, 2315)	1413	1374	1341	2.83	1.08	16	0.15
clqcolor-08-05-06	(116, 1114)	1114	1114	1114	107	62	1	0.007
dlx2_aa	(490, 2804)	1124	1020	970	3.12	0.94	32	0.023
addsub.boehm	(492, 1065)	1324	20256	347	35	29	> 657 000	time out

TAB. 3: L&S vs HYCAM : calcul de tous les MUS

- [4] R. Bruni. Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Applied Mathematics*, 130(2) :85–100, 2003.
- [5] R. Bruni. On exact selection of minimally unsatisfiable subformulae. *Annals of Mathematics and Artificial Intelligence*, 43(1) :35–50, 2005.
- [6] G. Davydov, I. Davydova, and H.K. Büning. An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF. *Annals of Mathematics and Artificial Intelligence*, 23(3–4) :229–245, 1998.
- [7] M. de la Banda, P. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets. In *Proceedings of the ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDL 2003)*, pages 32–43, 2003.
- [8] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactual. *Artificial Intelligence*, 57 :227–270, 1992.
- [9] N. Eén and N. Sörensson. Minisat home page <http://www.cs.chalmers.se/cs/research/formalmethods/minisat>, 2004.
- [10] H. Fleischner, O. Kullman, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoretical Computer Science*, 289(1) :503–516, 2002.
- [11] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'06)*, pages 387–391, Riva del Garda (Italy), 2006.
- [12] É. Grégoire, B. Mazure, and C. Piette. Tracking MUS and strict inconsistent cover. In *Proceedings of the Formal Methods in Computer Aided Design (FMCAD'06)*, pages 39–46, San Jose (USA), 2006.
- [13] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.
- [14] H. Kautz, B. Selman, and D. McAllester. Walksat in the SAT'2004 competition. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, 2004.
- [15] M.H. Liffiton and K.A. Sakallah. On finding all minimally unsatisfiable subformulas. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 173–186, 2005.
- [16] I. Lynce and J. Marques-Silva. On computing minimum unsatisfiable cores. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, 2004.
- [17] Y. Oh, M.N. Mneimneh, Z.S. Andraus, K.A. Sakallah, and I.L. Markov. AMUSE : a minimally-unsatisfiable subformula extractor. In *Proceedings of the Design Automation Conference (DAC'04)*, pages 518–523, 2004.
- [18] C. H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1) :2–13, 1988.
- [19] SATLIB. Benchmarks on SAT <http://www.satlib.org>.
- [20] L. Zhang. On subsumption removal and on-the-fly cnf simplification. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 482–489, 2005.
- [21] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, Portofino (Italy), 2003.