UNIVERSITÉ D'ARTOIS

SAT et au-delà de SAT : Modèles et Algorithmes

Habilitation à Diriger des Recherches

(Spécialité Informatique)

Université d'Artois

présentée et soutenue publiquement le 29 janvier 2010

par

Bertrand Mazure

Composition du jury

Rapporteurs :	Christian Bessière	Directeur de Recherche au CNRS, LIRMM, UMR 5506
	Felip Manyà	Directeur de Recherche au IIIA-CSIC, Barcelone
	Frédéric Saubion	Professeur à l'Université d'Angers
Examinateurs :	Éric Grégoire	Professeur à l'Université d'Artois
	Chu Min Li	Professeur à l'Université de Picardie
	Pierre Marquis	Professeur à l'Université d'Artois
Directeur :	Lakhdar Saïs	Professeur à l'Université d'Artois

Centre de Recherche en Informatique de Lens – CNRS UMR 8188
Université d'Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : Tél.: +33 (0)3 21 79 17 23 – Fax : +33 (0)3 21 79 17 70
http://www.cril.fr

Remerciements

Je tiens à remercier en premier lieu Christian BESSIÈRE, Felip MANYÀ et Frédéric SAUBION pour avoir accepté de rapporter ce travail, malgré leurs nombreuses activités et responsabilités. Mes remerciements vont également à Chu Min LI et à Pierre MARQUIS pour l'intérêt qu'ils ont porté à mon travail et pour avoir accepté de siéger dans ce jury en qualité d'examinateurs.

Éric GRÉGOIRE m'a engagé et dirigé (en tant que directeur de thèse puis en tant que directeur de laboratoire) sur la voie de la recherche depuis mon DEA. Je lui dois beaucoup et ces quelques lignes sont là pour l'en remercier.

Sont venues les lignes où je dois exprimer mes remerciements à celui qui a accepté de diriger cette HDR. Lakhdar SAÏS accompagne mes recherches depuis un samedi après-midi de janvier 95 où il m'a reçu dans son bureau du CRIL à l'IUT de Lens pour me présenter le sujet de stage de DEA qu'il proposait. Au travers notre collaboration étroite et quasi quotidienne, j'ai beaucoup appris et je continue à beaucoup apprendre. Il m'a transmis une part de son immense goût pour la recherche et il représente beaucoup plus qu'un simple responsable de thèse ou d'équipe. Déjà dans les remerciements de ma thèse, j'écrivais qu'une page n'aurait pas suffi à lui témoigner toute ma gratitude, je pense pouvoir surenchérir dans l'HDR en écrivant que ce manuscrit entier ne suffit pas non plus. Lakhdar, je t'exprime ma profonde amitié et je te dis simplement un immense merci !

Ces remerciements s'adressent également à celui qui partage le quotidien de mon bureau depuis de nombreuses années. Jean-Luc COQUIDÉ supporte le joyeux désordre que je laisse dans le bureau, mes goûts musicaux parfois particuliers, mais il m'est surtout d'un précieux soutien et sans ses nombreux encouragements cette HDR n'aurait pas vu le jour. Merci à toi Jean-Luc.

Merci aux collègues du CRIL et de la faculté des Sciences de Lens avec une mention spéciale pour Cédric à qui je souhaite une longue vie aux *filstool* et pour Nadine qui a géré seule durant de longues années les paperasses administratives de plus de cinquante personnes et particulièrement les miennes, n'étant pas très doué pour cette tâche.

Merci enfin à mes proches, à Valérie et à Ève que j'ai délaissées depuis quelque temps afin de boucler ce mémoire et à mes parents pour leur indéfectible soutien.

À Ève

Table des matières

Table des illustrations

Liste des figures

Liste des tables

Liste des algorithmes

Avant-propos et Organisation du mémoire

Ce mémoire a pour objectif de présenter les principaux travaux de recherche que j'ai réalisés depuis ma thèse. Ceux-ci ont été conduits au sein du CRIL et plus précisément au sein de l'axe « *Algorithmique pour l'inférence et la prise de décision* ». Il est donc naturel que nombre de ces travaux soient le fruit d'étroites collaborations avec Éric GRÉGOIRE directeur du CRIL et Lakhdar SAÏS responsable de cet axe. Rares sont les travaux de recherche qui s'accomplissent dans la solitude, aussi ai-je trouvé normal de les associer à ce manuscrit dès le début. Je désire également y associer l'ensemble des personnes avec qui j'ai eu la chance de travailler, discuter, échanger, ... au CRIL lors de réunions (le plus souvent informelles) ou lors de colloques et autres assemblées. Parmi ces personnes, une mention spéciale va aux doctorants que j'eus ou ai l'honneur de co-encadrer depuis presque dix ans : David ANSART, Richard OSTROWSKI, Olivier FOURDRINOY, Cédric PIETTE et Jean-Marie LAGNIEZ. Sans toutes ces personnes explicitement nommées ou juste évoquées, ce document n'aurait pas vu le jour et c'est pour elles que le terme *nous* sera employé tout au long de ce mémoire.

Avant de débuter la rédaction de ce manuscrit, j'ai naturellement feuilleté de nombreux mémoires d'habilitation à diriger des recherches. La grande majorité d'entre eux se décompose en trois parties : synthèse des travaux de recherche, curriculum vitae, sélection de publications. N'ayant pas trouvé de présentation plus originale, j'ai donc également adopté cette « trilogie ».

La première partie de ce mémoire est donc une synthèse de mes activités scientifiques. La forme et l'organisation de cette partie centrale du mémoire ont fait l'objet de longues hésitations et ont à maintes reprises été modifiées. La structuration retenue se veut somme toute naturelle et se décompose en quatre chapitres.

Le premier chapitre présente le contexte de mes travaux, décrit les problèmes abordés et définit les termes et les notations les plus utilisés dans ce mémoire. Les deux chapitres suivants traitent principalement de mes contributions : le premier développe mes travaux sur SAT et le second mes travaux sur des problèmes connexes à SAT. Sur SAT, après un bref état de l'art, je décris mes travaux autour de trois thèmes : l'hybridation de solveurs, l'exploitation de propriétés structurelles et les techniques de simplification. Le chapitre intitulé « Au-delà de SAT » regroupe mes contributions sur des problèmes tels que la recherche de noyaux incohérents dans le cadre propositionnel et dans le cadre plus général de réseaux

de contraintes, la résolution de problèmes exprimés à l'aide de logiques non monotones et du premier ordre, la compilation de bases de connaissances, la résolution des formules booléennes quantifiées. Dans le dernier chapitre, j'énonce quelques perspectives de recherche.

Cette organisation a largement inspiré le titre du mémoire. Cependant, elle ne permet pas toujours de bien mettre en exergue les contributions par rapport à l'état de l'art. À cette fin, les paragraphes traitant exclusivement de mes contributions sont signalés par un trait vertical dans la marge, comme illustré sur ce paragraphe. De plus, les références à des publications où je suis auteur ou co-auteur apparaissent en caractère gras dans le texte.

La seconde partie du manuscrit est constituée de mon curriculum vitae. Il se décompose en cinq parties détaillant respectivement :

1. mon état civil et mes parcours universitaires et professionnels ;
2. mes activités liées à la recherche ;
3. celles liées à l'enseignement ;
4. mes responsabilités administratives ;
5. ma liste de publications.

La troisième et dernière partie du mémoire regroupe une sélection de publications qui me semblent les plus significatives et les plus représentatives de mes travaux de recherche. Le choix a été fait de manière à couvrir les différents aspects abordés dans la première partie de ce manuscrit.

Partie I

Synthèse des travaux de recherche

Sommaire

Chapitre 1

Problématique et Notations

La complexité croissante des systèmes et des problèmes qu'engendre le développement des nouvelles technologies, pose au monde industriel, économique et académique de nouveaux défis en termes de prise de décision, de planification et d'organisation. Dans ce contexte, le caractère combinatoire et complexe de nombreux problèmes posés rend les approches issues de l'intelligence artificielle particulièrement prometteuses pour répondre à ces nouveaux challenges. En effet, face à cette complexité croissante des problèmes, un objectif majeur demeure la mise en œuvre de systèmes simples, génériques et efficaces.

La logique propositionnelle offre un cadre propice à cette mise en œuvre. En effet, de nombreux problèmes pratiques peuvent se représenter de manière simple en logique propositionnelle. Il reste que la déduction en calcul propositionnel n'admet, à l'heure actuelle, aucun algorithme général et efficace dans tous les cas. En effet, elle repose directement sur le test de cohérence d'une formule propositionnelle[1], c'est-à-dire sur SAT (problème NP-complet de référence). S'agissant de la résolution de problèmes NP-complets, le défi pour la communauté scientifique est de repousser aussi loin que possible la taille des instances qui peuvent être résolues en pratique. Mes travaux qui concernent principalement l'algorithmique pour SAT et la résolution de problèmes autour de celui-ci, s'inscrivent pleinement dans cet objectif.

L'objet de ce chapitre n'est pas de réaliser un état de l'art sur ce domaine mais juste de définir formellement les problèmes adressés dans ce mémoire et d'introduire les notations utilisées dans le reste du manuscrit. En effet, vu le nombre important de travaux de recherche réalisés depuis de nombreuses années sur SAT, il serait prétentieux de vouloir réaliser un inventaire complet. Ceci dépasserait largement le cadre de cette synthèse. Nous nous limitons à un exposé des travaux importants et/ou en relation directe

[1] $\mathcal{F} \models f$ si et seulement si $(\mathcal{F} \wedge \neg f)$ n'est pas satisfiable.

avec nos travaux de recherche. Pour un état de l'art plus complet sur le domaine, le lecteur intéressé pourra se référer à deux ouvrages récents : un en français [Saï08], l'autre en anglais [BHvMW09].

1.1 SAT

Une *variable propositionnelle* ou *booléenne* est une variable qui prend des valeurs de vérité. En logique classique, ces valeurs de vérité sont réduites à deux valeurs qui peuvent être représentées par $\{\mathsf{faux}, \mathsf{vrai}\}$, $\{0, 1\}$ ou encore $\{\bot, \top\}$. On désigne par *littéral* une variable propositionnelle ou sa négation. Pour une variable x, x représente le littéral *positif* et $\neg x$ le littéral *négatif*. Pour un littéral ℓ, son littéral complémentaire ou opposé est noté $\overline{\ell}$.

Une *clause* est une disjonction finie de littéraux. Nous représentons indifféremment une clause par la disjonction de ses littéraux $(\ell_1 \vee \ell_2 \vee ... \vee \ell_n)$ ou par l'ensemble de ses littéraux $\{\ell_1, \ell_2, ..., \ell_n\}$.

Une formule propositionnelle mise sous *forme normale conjonctive* (CNF) est une conjonction finie de clauses. Comme pour les clauses, une CNF peut se noter soit par la conjonction de ses clauses $(\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n)$, soit par l'ensemble de ses clauses $\{\alpha_1, \alpha_2, ..., \alpha_n\}$.

Une *affectation* ou *interprétation* I d'un ensemble X de variables propositionnelles est une fonction $I : X \longrightarrow \{\mathsf{faux}, \mathsf{vrai}\}$. Lorsqu'elle n'est définie que pour un sous-ensemble des variables, elle est dite partielle ou incomplète. Un littéral ℓ (respectivement $\neg\ell$) de X est vrai dans I si et seulement si $I(\ell) = \mathsf{vrai}$ (respectivement $I(\ell) = \mathsf{faux}$). Une interprétation se représente par l'ensemble[2] des littéraux vrais. Une clause α est satisfaite par une interprétation I si et seulement si $\alpha \cap I \neq \emptyset$. Une CNF Σ est satisfaite par une interprétation I si et seulement si I satisfait toutes les clauses de Σ (c'est-à-dire $\forall \alpha \in \Sigma, \alpha \cap I \neq \emptyset$). La CNF est dite alors *satisfiable*[3] et I est appelée une solution ou un *modèle* de Σ.

SAT (forme abrégée du problème de satisfiabilité[4] d'une formule propositionnelle mise sous normale conjonctive) est un problème de décision qui consiste à déterminer si une CNF admet ou non au moins un modèle. Ce problème constitue un problème fondamental à la fois théoriquement et pratiquement. Théoriquement car il occupe un rôle central en théorie de la complexité où il représente le problème NP-complet de référence [Coo71]. Pratiquement, car de nombreux problèmes en informatique s'y ramènent ou le contiennent naturellement (démonstration automatique, base de données déductives, VLSI, etc.).

Exemple 1. Soit une CNF $\Sigma = (b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b) \wedge (\neg c \vee \neg a)$. Σ est satisfiable et admet $M = \{a, b, \neg c\}$ comme modèle.

Soit $\Sigma' = \Sigma \wedge (\neg a \vee \neg b \vee c)$. Σ' n'admet aucun modèle, on dit alors que Σ' est non satisfiable[5].

1.2 Définitions de base et notations

Quelques définitions fondamentales et notations utilisées dans le reste du document sont introduites dans cette partie. Pour plus de clarté, nous avons préféré les regrouper dans cette section. Les notations spécifiques sont introduites au moment de leur utilisation.

[2] Afin de lever les ambiguïtés que pourrait causer l'utilisation des différentes notations ensemblistes, nous utilisons des lettres latines minuscules pour désigner des variables ou des littéraux, des lettres latines majuscules pour les interprétations, des lettres grecques minuscules pour les clauses et des lettres grecques majuscules pour les CNFs.

[3] Le terme « *satisfiable* » est un anglicisme couramment utilisé dans la littérature traitant de SAT. Néanmoins, on trouve dorénavant sa définition sur certains dictionnaires en ligne (`www.mediadico.com`) : « satisfiable : (adjectif) terme qualifiant une fonction propositionnelle vraie dans certains cas déterminés ».

[4] Nouvel anglicisme, pour lequel une définition peut être trouvée sur le même site, mais également sur le site `www.larousse.fr` : « satisfiabilité : (nom féminin) caractère d'une proposition satisfiable ».

[5] L'anglicisme « *unsatisfiable* » est parfois employé dans la littérature pour qualifier une instance non satisfiable. On trouve même également sa version francisée avec l'orthographe « *insatisfiable* ». N'ayant pas trouvé de définitions pour ces termes dans les différents dictionnaires testés, nous évitons leur emploi dans ce document.

ALGORITHME 1.1 : `UnitPropagation()`

input : Σ une CNF
output : Σ^* la formule Σ close par propagation unitaire

```
1 begin
2   while ((∃{ℓ} ∈ Σ) and (⊥ ∉ Σ)) do
3     Σ ⟵ Σ|ℓ ;
      /* Supprime de Σ toutes les clauses contenant ℓ                */
      /* et toutes les occurrences ℓ̄ dans les clauses restantes      */
4   end
5   return Σ;
6 end
```

1.2.1 Clauses : propriétés

Une clause α est dite *unitaire* si elle est réduite à un seul littéral, c'est-à-dire $|\alpha| = 1$[6]. L'unique littéral composant une clause unitaire est qualifié de *littéral unitaire*. De la même manière, une *clause binaire* est de taille deux, c'est-à-dire $|\alpha| = 2$ et plus généralement une clause de taille n est qualifiée de clause n*-aire*. La *clause vide* ne peut être satisfaite, elle est notée $\bot$ et elle s'assimile à la valeur de vérité faux.

Une clause α est dite *unisatisfaite* pour une interprétation I si et seulement si $|I \cap \alpha| = 1$.

Une clause est dite de *Horn* (respectivement *reverse-Horn*) si et seulement si elle possède au plus un littéral positif (respectivement négatif). Une clause est dite *positive* (respectivement *négative*), si et seulement si tous ses littéraux sont positifs (respectivement négatifs).

Une clause α *subsume* une clause β si et seulement si $\alpha \subseteq \beta$.

Deux clauses α et β se *résolvent* si et seulement si il existe un littéral ℓ tel que $\ell \in \alpha$ et $\overline{\ell} \in \beta$. La clause $(\alpha \setminus \{\ell\}) \cup (\beta \setminus \{\overline{\ell}\})$ est appelée *résolvante* en ℓ de α et β, nous la notons $\biguplus_\ell[\alpha, \beta]$.

Une clause α est dite *tautologique*, c'est-à-dire universellement satisfaite, si et seulement si il existe une variable x telle que $x \in \alpha$ et $\neg x \in \alpha$.

1.2.2 Simplification d'une CNF

Pour une CNF Σ, $\mathcal{V}_\Sigma$ désigne l'ensemble des variables propositionnelles de Σ. L'ensemble des littéraux apparaissant dans Σ est noté $\mathcal{L}_\Sigma$ ($|\mathcal{L}_\Sigma| \leq 2 \times |\mathcal{V}_\Sigma|$).

Soient Σ une CNF et ℓ un littéral de $\mathcal{L}_\Sigma$, nous notons $\Sigma|_\ell$ l'ensemble de clauses Σ simplifié par l'affectation du littéral ℓ à vrai, $\Sigma|_\ell = \{\alpha \setminus \{\overline{\ell}\} | \alpha \in \Sigma \text{ et } \ell \notin \alpha\}$. Par extension, nous notons $\Sigma|_{\{\ell_1,\ell_2,...\ell_n\}} = (...((\Sigma|_{\ell_1})|_{\ell_2})...|_{\ell_n})$ la CNF Σ simplifiée par l'ensemble de littéraux $\{\ell_1, \ell_2, ..., \ell_n\}$.

Soit ℓ un littéral unitaire de Σ, c'est-à-dire $\{\ell\} \in \Sigma$. Σ est satisfiable si et seulement si $\Sigma|_\ell$ est satisfiable. Cette propriété permet de simplifier la CNF dès lors qu'elle possède une clause unitaire. Cette simplification est appelée *propagation unitaire*. Nous notons Σ^* la CNF close par propagation unitaire.

Un littéral ℓ est dit *monotone* ou *pur* pour une CNF Σ si et seulement si $\ell \in \mathcal{L}_\Sigma$ et $\overline{\ell} \notin \mathcal{L}_\Sigma$. Autrement dit, le littéral ℓ apparaît uniquement positivement (ou négativement) dans les clauses de Σ. À l'instar des littéraux unitaires, si ℓ est pur pour Σ, alors Σ est satisfiable si et seulement si $\Sigma|_\ell$ est satisfiable.

[6] La taille d'une clause correspond à la taille de l'ensemble des littéraux de la clause, c'est-à-dire au nombre de littéraux de la clause. De la même manière, la taille d'une CNF Σ, notée $|\Sigma|$, représente la taille de l'ensemble des clauses de Σ, c'est-à-dire le nombre de clauses de Σ.

ALGORITHME 1.2 : `PurePropagation()`

input : Σ une CNF
output : $\Sigma^{\square}$ la formule Σ close par propagation des littéraux purs

```
1 begin
2     while (∃ℓ ∈ L_Σ t.q. ℓ̄ ∉ L_Σ) do
3         Σ ⟵ Σ|_ℓ ;
          /* Supprime de Σ toutes les clauses contenant ℓ */
4     end
5     return Σ;
6 end
```

Cette simplification est nommée *propagation des littéraux purs*. Nous notons $\Sigma^{\square}$ la CNF Σ close par propagation des littéraux purs et $\Sigma^{\circledast}$ celle close à la fois par la propagation des littéraux unitaires et purs.

Les algorithmes 1.1 et 1.2 décrivent respectivement l'algorithme de propagation unitaire et propagation des littéraux purs.

1.2.3 Conséquence logique et restriction

La conséquence logique ($\models$) et SAT sont naturellement liés. En effet une formule Φ est une conséquence logique de Σ (noté $\Sigma \models \Phi$) si et seulement si tout modèle de Σ est un modèle Φ. Autrement dit, pour vérifier que Φ est une conséquence logique de Σ, il suffit de s'assurer que la formule $(\Sigma \wedge \neg\Phi)$ n'admet pas de modèle. Dans le cas, où Σ et Φ sont des CNF, cela revient à résoudre $|\Phi|$ instances de SAT : $\Sigma \models \Phi \iff (\forall \alpha = (\ell_1 \vee \ell_2 \vee ... \vee \ell_n) \in \Phi, (\Sigma \wedge (\overline{l_1}) \wedge (\overline{l_2}) \wedge ... \wedge (\overline{l_n}))$ n'admet pas de modèle). Lorsqu'une formule Σ n'admet pas de modèle nous notons $\Sigma \models \bot$.

La conséquence logique restreinte à la propagation unitaire, notée $\models^*$, limite le processus de déduction à la propagation unitaire. Formellement, pour deux CNF Σ et Φ, $\Sigma \models^* \Phi \iff (\forall \alpha = (\ell_1 \vee \ell_2 \vee ... \vee \ell_n) \in \Phi, (\Sigma|^*_{\{\overline{l_1},\overline{l_2},...,\overline{l_n}\}}$ contient la clause vide). La propagation unitaire d'une formule Σ nécessite un algorithme linéaire en temps ($\mathcal{O}(|\Sigma|)$), par conséquent $\Sigma \models^* \Phi$ peut être calculée en un temps polynomial ($\mathcal{O}(|\Phi| \times (|\Sigma| + \epsilon)$ où ϵ est la taille de la plus longue clause de Φ).

1.3 Quelques problèmes gravitant autour de SAT

L'objectif de cette partie du chapitre n'est pas de lister exhaustivement les problèmes liés directement à la résolution de SAT, la liste serait bien trop longue, mais de présenter des problèmes sur lesquels nous avons travaillé et qui sont donc évoqués dans les prochains chapitres.

1.3.1 MAXSAT et #SAT

De nombreux problèmes dérivent directement de SAT, les plus célèbres sont certainement les problèmes MAXSAT et #SAT qui sont respectivement les problèmes d'optimisation et de comptage associés à SAT. Formellement, MAXSAT est le problème de décision consistant à déterminer, pour une CNF Σ et un entier naturel k donnés, s'il existe une interprétation qui satisfait au moins k clauses de Σ. En pratique, le problème MAXSAT consiste à déterminer le plus grand k, c'est-à-dire trouver l'interprétation satisfaisant le plus de clauses. #SAT quant à lui est un problème de dénombrement qui consiste à déterminer le nombre de modèles d'une CNF.

1.3.2 MUS et MSS

Lorsqu'une CNF n'admet pas de modèle, la non-satisfiabilité de la CNF peut être due à un sous-ensemble de clauses. La recherche d'un MUS (« *Minimally Unsatisfiable Subformula* ») fournit une explication minimale (en terme du nombre de clauses nécessaires) de la non-satisfiabilité de la formule. Formellement, un MUS Γ d'une CNF Σ est un ensemble de clauses tel que :

- $\Gamma \subseteq \Sigma$;
- Γ n'admet pas de modèle ;
- $\forall \alpha \in \Gamma, \Gamma \setminus \{\alpha\}$ est satisfiable.

Exemple 2. Soit $\Sigma = (a) \wedge (\neg c) \wedge (\neg b \vee \neg a) \wedge (b) \wedge (\neg b \vee c)$. Σ possède deux MUS : $\Gamma_1 = \{a, b, \neg b \vee \neg a\}$ et $\Gamma_2 = \{\neg c, b, \neg b \vee c\}$

Une CNF peut posséder un nombre exponentiel de MUS. En effet, une CNF de n clauses peut contenir jusqu'à $C_n^{\frac{n}{2}}$ MUS dans le pire cas. Vérifier si un ensemble de clauses est un MUS ou non appartient à la classe de complexité DP-complet[7] [PW88] et vérifier si un ensemble de clauses fait partie de l'ensemble des MUS d'une CNF appartient à la classe de complexité $\sum_2^p$-difficile[8] [EG92].

Le concept dual aux MUS est la notion de MSS (« *Maximal Satisfiable Subformula* ») d'une CNF. Formellement, un MSS Ω d'une CNF Σ est un ensemble de clauses tel que :

- $\Omega \subseteq \Sigma$;
- Ω est satisfiable ;
- $\forall \alpha \in \Sigma \setminus \Omega, \Omega \cup \{\alpha\}$ n'admet pas de modèle.

Les ensembles complets de MUS et MSS d'une CNF se déduisent naturellement l'un de l'autre. En effet, le calcul des ensembles intersectants minimaux[9] de l'ensemble des MUS fournit l'ensemble des CoMSS de la CNF. Un CoMSS représente l'ensemble complémentaire de clauses d'un MSS dans la CNF.

Exemple 3. Si l'on considère l'ensemble de clauses Σ défini dans l'exemple 2, il admet cinq MSS : $\{\neg b \vee \neg a, b, \neg b \vee c\}$, $\{\neg c, \neg b \vee \neg a, b\}$, $\{a, \neg c, \neg b \vee \neg a, \neg b \vee c\}$, $\{a, b, \neg b \vee c\}$ et $\{a, \neg c, b\}$.
Et donc cinq CoMSS : $\{a, \neg c\}$, $\{a, \neg b \vee c\}$, $\{b\}$, $\{\neg b \vee \neg a, \neg c\}$ et $\{\neg b \vee \neg a, \neg b \vee c\}$.

Les problèmes MAXSAT et de recherche de MUS ou de MSS d'une instance de SAT sont évidemment corrélés puisque la valeur cherchée pour MAXSAT correspond à la taille du plus grand MSS de l'instance.

1.3.3 QBF

Le problème QBF (« *Quantified Boolean Formula* ») est une généralisation de SAT dans laquelle un *quantificateur* (existentiel ($\exists$) ou universel ($\forall$)) peut être appliqué à chaque variable de la formule. Nous parlons de généralisation car une instance de SAT peut être vue comme une instance de QBF où toutes les variables sont quantifiées de manière existentielle.

Une formule QBF est dite sous *forme normale prénexe* si et seulement si elle est de la forme $\mathcal{Q}_1 x_1, \mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi$ avec $\mathcal{Q}_i \in \{\forall, \exists\}$ et Ψ une CNF. Ψ est appelée la *matrice* et $\mathcal{Q}_1 x_1, \mathcal{Q}_2 x_2, ... \mathcal{Q}_n x_n$ le *préfixe*. La formule suivante est une formule QBF sous forme normale prénexe : $\forall x \exists y \exists z (x \vee \neg y) \wedge (z)$.

[7] Un problème $\mathcal{P}$ appartient à la classe de complexité DP s'il peut être écrit $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$ avec $\mathcal{P}_1 \in$ NP et $\mathcal{P}_2 \in$ CoNP.

[8] Ensemble des problèmes appartenant à NP à condition de disposer d'un oracle lui-même NP.

[9] Soit $\mathcal{S}$ un ensemble d'ensembles, $\mathcal{S}_\cap$ est un ensemble intersectant, « *hitting set* » en anglais, de $\mathcal{S}$ si et seulement si il contient au moins un élément de chacun des ensembles de $\mathcal{S}$: $\forall s \in \mathcal{S}, s \cap \mathcal{S}_\cap \neq \emptyset$. Il est minimal si et seulement si il n'existe pas d'ensemble intersectant $\mathcal{S}'_\cap$ de $\mathcal{S}$ tel que $\mathcal{S}'_\cap \subset \mathcal{S}_\cap$. Ce problème est équivalent au calcul des transversaux minimaux pour un hypergraphe. Il a été largement étudié ainsi que ses applications à la logique et à l'intelligence artificielle [EG02].

Toute formule QBF peut se ramener à une forme normale prénexe, nous nous limitons donc à cette représentation. Il est d'usage de regrouper dans des ensembles les variables contiguës de même quantificateur. La QBF s'écrit alors : $\mathcal{Q}_1 X_1, \mathcal{Q}_2 X_2, ... \mathcal{Q}_k X_k \Psi$ avec $\forall i \neq j \in [1..k], X_i \subseteq \mathcal{V}_\Psi$, $X_i \cap X_j = \emptyset$ et $\bigcup_{i \in [1..k]} X_i = \mathcal{V}_\Psi$.

Le problème QBF est un problème de décision qui consiste à déterminer si une telle formule QBF sous forme normale prénexe est valide. La validité d'une QBF $\mathcal{Q}_1 x_1, \mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi$ se définit récursivement de la manière suivante :

- si le préfixe est vide alors la QBF est valide si et seulement si la matrice est satisfiable ;
- sinon :
  - si $\mathcal{Q}_1 = \forall$ alors la QBF est valide si et seulement si les deux QBF $\mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi|_{x_1}$ et $\mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi|_{\neg x_1}$ sont valides ;
  - si $\mathcal{Q}_1 = \exists$ alors la QBF est valide si et seulement si au moins l'une des deux QBF $\mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi|_{x_1}$ ou $\mathcal{Q}_2 x_2, ..., \mathcal{Q}_n x_n \Psi|_{\neg x_1}$ est valide.

Exemple 4. La formule QBF $\forall x \exists y \exists z (x \vee \neg y) \wedge (z)$ est valide, alors que $\exists x \exists y \forall z (x \vee \neg y) \wedge (z)$ ne l'est pas.

À l'instar de SAT qui représente le problème de référence pour la classe de complexité NP-complet, le problème QBF est le problème canonique pour la classe PSPACE-complet[10]. Cela sous-entend que ni la validité d'une QBF, ni même l'exactitude d'une solution donnée ne peuvent être vérifiées en temps polynomial via un algorithme déterministe. D'ailleurs, représenter de manière succincte les solutions d'une instance QBF demeure un problème pour lequel nous n'avons pas de solution satisfaisante.

1.3.4 CSP

SAT s'inscrit dans le cadre plus général des problèmes de satisfaction de contraintes. Un problème de satisfaction de contraintes (CSP, « *Constraint Satisfaction Problem* ») est un couple $P = \langle \mathcal{V}, \mathcal{C} \rangle$ où :

1. $\mathcal{V}$ est un ensemble fini de n variables $\{v_1, \ldots, v_n\}$ tel que chaque variable $v_i \in \mathcal{V}$ possède un *domaine* noté $dom(v_i)$, qui contient l'ensemble des valeurs possibles de v_i ;
2. $\mathcal{C}$ est un ensemble fini de m *contraintes* $\{c_1, \ldots, c_m\}$ tel que chaque contrainte $c_j \in \mathcal{C}$ porte sur un sous-ensemble de variables de $\mathcal{V}$ que l'on note $scp(c_j)$.

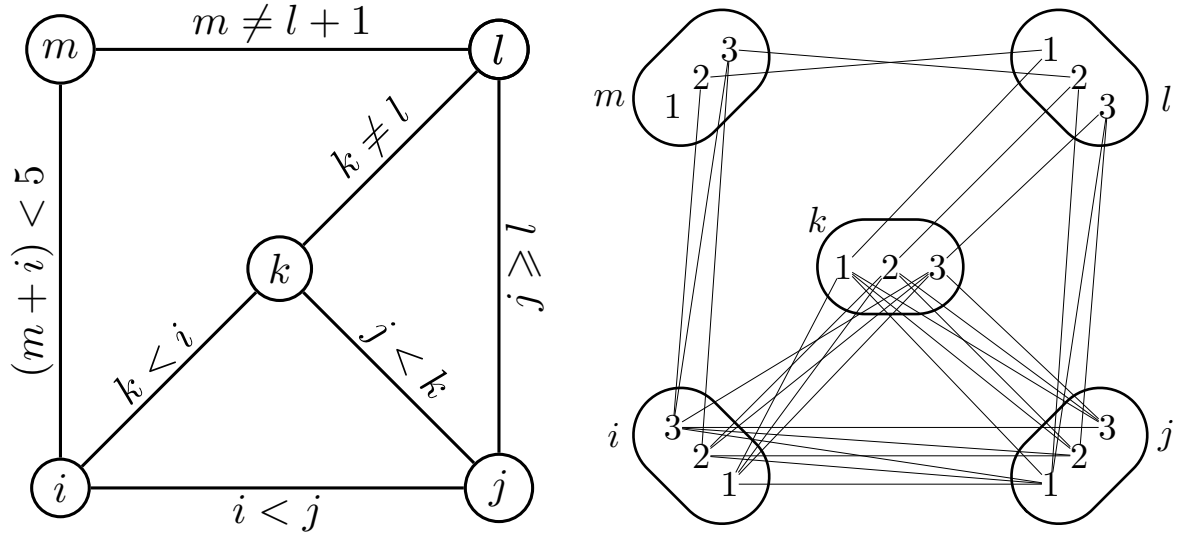
Une contrainte c_j se définit soit à l'aide d'une *relation de compatibilité*, notée $\mathcal{R}(c_j)$, qui contient l'ensemble des *tuples de valeurs autorisées*, soit à l'aide de la relation duale que nous notons $\overline{\mathcal{R}}(c_j)$ et qui contient l'ensemble des *tuples de valeurs interdites*.

Exemple 5. Soit la contrainte mathématique $x < y - 1$ avec $dom(x) = \{0, 1\}$, $dom(y) = \{1, 2, 3\}$: $scp(x < y - 1) = \{x, y\}$, $\mathcal{R}(x < y - 1) = \{(x = 0, y = 2), (x = 0, y = 3), (x = 1, y = 3)\}$ et de manière duale $\overline{\mathcal{R}}(x < y - 1) = \{(x = 0, y = 1), (x = 1, y = 1), (x = 1, y = 2)\}$.

Dans le cas particulier de SAT, $\mathcal{V}$ est égal à l'ensemble des variables propositionnelles de la CNF, les variables possèdent toutes le même domaine : $\{\mathsf{faux}, \mathsf{vrai}\}$, $\mathcal{C}$ correspond à l'ensemble des clauses de la CNF, la portée de la contrainte c est égale à l'ensemble des variables de la clause c, sa relation d'incompatibilité ne contient qu'un seul tuple de valeurs $\overline{\mathcal{R}}(c) = \{(\forall x \in scp(c), x = \mathsf{faux}$ si x est positif, vrai sinon$)\}$.

[10] PSPACE représente l'ensemble des problèmes qui peuvent être résolus sur une machine de TURING en utilisant un espace polynomial en fonction de la taille de l'entrée.

FIGURE 1.1 : Représentations graphiques d'un CSP binaire



(a) Réseau de contraintes

(b) Graphe de micro-structure

Résoudre un CSP $P = \langle \mathcal{V}, \mathcal{C} \rangle$ consiste à vérifier si P admet au moins une solution, c'est-à-dire une affectation de valeur pour chaque variable de $\mathcal{V}$ telle que chaque contrainte de $\mathcal{C}$ est satisfaite. Si P admet au moins une solution, alors il est dit satisfiable.

Tout CSP peut s'écrire sous la forme d'un CSP *binaire*, c'est-à-dire un CSP où chaque contrainte ne porte que sur deux variables. Un CSP binaire est souvent représenté par son *réseau de contraintes*. Ce réseau de contraintes est un graphe non orienté où les nœuds sont les variables et où il existe une arête entre deux variables si et seulement si elles appartiennent à la portée d'une même contrainte. Les arêtes sont étiquetés par les contraintes. Mais il peut être également utile de représenter un CSP par son *graphe de micro-structure*, qui est un graphe n-parti (pour un CSP à n variables) regroupant au sein de chaque « partie » les valeurs du domaine de chaque variable et reliant par une arête chaque couple interdit (ou autorisé) de valeurs par les contraintes du problème. Dans les graphes de micro-structure présentés dans ce document, nous choisissons de représenter les tuples interdits du CSP.

Exemple 6. Soit $\mathcal{V} = \{i, j, k, l, m\}$ où chaque variable possède le même domaine $\{1, 2, 3\}$. Soit $\mathcal{C}$ l'ensemble des sept contraintes mathématiques suivantes : $\mathcal{C} = \{(i < j), (j < k), (k < i), (k \neq l), (j \geq l), (m \neq l + 1), (m + i < 5)\}$. Les représentations graphiques du CSP $P = \langle \mathcal{V}, \mathcal{C} \rangle$ sont illustrées dans la figure 1.1.

Lorsqu'un CSP n'admet pas de solution, il possède au moins un MUC « *Minimally Unsatisfiable Core* ». Un MUC est un sous-ensemble de contraintes d'un CSP non satisfiable et tel que tous ses sous-ensembles propres soient satisfiables. C'est la transposition des MUS du cadre SAT aux problèmes CSP. Formellement, soient $P = \langle \mathcal{V}, \mathcal{C} \rangle$ et $P' = \langle \mathcal{V}' \subseteq \mathcal{V}, \mathcal{C}' \subseteq \mathcal{C} \rangle$ deux CSP. P' est un MUC de P si et seulement si :

- P' n'admet pas de solution ;
- pour tout CSP $P'' = \langle \mathcal{V}'' \subseteq \mathcal{V}', \mathcal{C}'' \subset \mathcal{C}' \rangle$, P'' est satisfiable.

Exemple 7. Dans l'exemple précédent, P n'admet pas de solution et il contient un MUC $P' = \langle \{i, j, k\}, \{(i < j), (j < k), (k < i)\} \rangle$.

Tout comme dans le cadre de SAT, extraire un MUC d'un CSP est un problème NP-difficile.

Sommaire

Chapitre 2

SAT

SAT a fait l'objet de nombreuses recherches durant ces vingt dernières années, trop nombreuses pour avoir la prétention d'en faire l'inventaire complet au travers de ce mémoire. Aussi, en plus de la présentation de nos travaux, nous présentons uniquement quelques résultats importants qui ont influencé la communauté et en particulier nos recherches ces dernières années.

Nos travaux ont essentiellement porté sur la proposition de nouveaux modèles et/ou algorithmes pour SAT dans le but d'étendre la classe des problèmes traitables en pratique. Depuis la thèse de doctorat, nous pouvons distinguer trois grandes orientations dans nos travaux concernant la résolution pratique de d'instances de SAT :

1. la mise au point d'un algorithme hybride, c'est-à-dire combinant recherche locale et algorithme complet énumératif ;
2. la détection et l'exploitation de propriétés structurelles ;
3. la proposition de nouvelles techniques de simplification.

C'est donc naturellement que ce chapitre se structure en trois parties principales, chacune traitant un aspect énoncé ci-dessus.

2.1 Hybridation de solveurs

De nombreux algorithmes ont été proposés pour résoudre SAT. Les principes à la base de ces méthodes sont également variés : résolution [Rob65, KK71, Lov78], réécriture [Boo54, Sha40], énumération [DLL62, JW90, MSS96, MMZ+01], comptage du nombre de solutions [Iwa89], recherche locale [SLM92], programmation linéaire en nombre entiers [BJL86, Hoo88, BP92, BP93], diagrammes binaires

de décision [Ake78, Bry92, US94], algorithmes génétiques et évolutionnistes [Hao95, GMR02, LSH06], etc.

En pratique afin de déterminer si une CNF admet ou non un modèle, deux de ces principes sont plus usités que les autres car ils s'avèrent plus efficaces sur un grand nombre d'instances. Ces deux grandes familles d'algorithmes sont : les algorithmes énumératifs reposant le plus souvent sur la célèbre procédure de Martin DAVIS, Hilary PUTNAM, George LOGEMANN et Donald LOVELAND [DLL62], abrégée en DPLL et les algorithmes de recherche locale. Ces deux types de méthodes s'opposent sur bien des points : l'une est *complète*[11] l'autre non ; l'une est capable de résoudre des instances satisfiables aléatoires de grande taille alors que l'autre peine à résoudre des instances de 500 variables de ce type ; l'une se base sur une exploration systématique de l'espace de recherche, l'autre sur une exploration stochastique ; etc. L'idée de combiner ces approches semble donc naturelle afin d'exploiter les forces de l'une pour atténuer les faiblesses de l'autre, le but étant d'obtenir un algorithme au moins aussi efficace que les deux approches prises séparément. La réalisation d'un tel algorithme est un des dix challenges proposés à la communauté par Bart SELMAN *et al.* en 1997 [SKM97] :

« ***CHALLENGE 7 :*** *Demonstrate the succesful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.* »

En 2003, Henri KAUTZ et Bart SELMAN dressaient un bilan des avancées réalisées sur ces dix challenges [KS03] et constataient que ce challenge restait ouvert.

Nous avons proposé différents modèles d'hybridation et développé plusieurs algorithmes suivant ces modèles. Nous les détaillons ci-après (voir § 2.1.3). Auparavant nous faisons un rapide tour d'horizon des méthodes utilisées pour l'hybridation, à savoir DPLL ainsi que ses variantes modernes de type CDCL et les méthodes de recherche locale.

2.1.1 L'algorithme DPLL de 1962 à nos jours

L'histoire de l'algorithme DPLL [DLL62] ne commence pas en 1962 mais en 1960. En effet, cet algorithme copie une partie du modèle algorithmique proposé dans l'algorithme de *résolution dirigée* par Martin DAVIS et Hilary PUTNAM [DP60], communément appelé DP. DPLL emprunte à DP :

- ses critères d'arrêt :
  - l'absence de clause indiquant que l'instance est satisfiable ;
  - l'apparition de la clause vide signifiant que l'instance n'admet pas de modèle ;
- ses simplifications par les littéraux unitaires et purs ;
- le fait de réduire d'une variable la CNF à chaque étape, permettant ainsi de réduire la complexité du problème à traiter petit à petit.

Ces emprunts expliquent certainement pourquoi les algorithmes DP et DPLL ont souvent été confondus dans la littérature et également pourquoi le nom de Hilary PUTNAM est également associé à DPLL alors qu'il ne figure pas dans la liste des auteurs de l'article introduisant DPLL [DLL62].

La différence entre les deux algorithmes se situe dans la manière d'éliminer les variables. Dans DP, une variable est supprimée en générant toutes les résolvantes qu'il est possible de construire en sélectionnant cette variable comme pivot de la résolvante. Ceci permet de supprimer toutes les clauses contenant un littéral issu de cette variable. L'inconvénient majeur de l'algorithme DP est sa complexité spatiale exponentielle due à cette génération de résolvantes. Pour y remédier, la procédure DPLL se

[11] Une méthode est dite complète pour SAT si et seulement si elle est capable de décider en un temps fini de la satisfiabilité de l'instance, que celle-ci soit satisfiable ou non. Ces algorithmes sont exponentiels dans le pire cas. À l'inverse, les méthodes incomplètes sont, le plus souvent, incapables de prouver qu'une instance n'admet pas de modèle mais peuvent uniquement prouver qu'une instance est satisfiable. Ces algorithmes s'exécutent le plus souvent en temps polynomial par rapport à la taille de la CNF.

ALGORITHME 2.1 : DPLL

input : Σ une CNF
output : vrai si Σ est satisfiable, faux sinon

```
1 begin
2     Σ* ⟵ UnitPropagation(Σ);
3     if (⊥ ∈ Σ*) then return faux;
4     Σ⊛ ⟵ PurePropagation(Σ*);
5     if (Σ⊛ = ∅) then return vrai;
6     v ⟵ DecisionHeuristic(Σ⊛);
7     if (DPLL(Σ⊛|v)) then return vrai;
8     else return DPLL(Σ⊛|¬v);
9 end
```

fonde sur le principe de *séparation*. L'idée est qu'une formule Σ est satisfiable si et seulement si $\Sigma|_v$ est satisfiable ou $\Sigma|_{\neg v}$ est satisfiable, $\forall v \in \mathcal{V}_\Sigma$. Il s'agit donc d'un algorithme de recherche en profondeur d'abord dans un arbre binaire où chaque nœud correspond à une sous-formule. Les feuilles correspondent aux critères d'arrêt de l'algorithme DP, à savoir une formule vide indiquant que toutes les clauses sont satisfaites, ou une formule contenant la clause vide indiquant une contradiction et l'absence de modèle dans cette branche de l'arbre. Si l'algorithme termine en ayant décrit un arbre possédant uniquement des feuilles contenant la clause vide, alors la CNF n'admet pas de modèle. L'intérêt de cette méthode réside principalement dans sa complexité spatiale qui n'est pas exponentielle, comme dans la procédure DP, mais polynomiale. Encore aujourd'hui, de nombreux solveurs (*satz* [LA97], *kcnfs* [DD04], *eqsatz* [Li00], *grasp* [MSS96], *chaff* [MMZ+01], *minisat* [ES04], etc.) utilisent ce modèle algorithmique.

Les algorithmes DP et DPLL partagent également le fait de simplifier la formule par les littéraux unitaires et purs à chaque étape. Si la simplification par les littéraux unitaires est toujours réalisée car elle permet des coupes substantielles dans l'arbre de recherche, celle des littéraux purs est moins utilisée dans les solveurs modernes qui utilisent des structures de données (voir § Les structures de données paresseuses) ne leur permettant pas de détecter aisément qu'un littéral est pur.

Un point crucial quant à l'efficacité de la procédure DPLL est le choix de la variable pour la règle de séparation. Ce choix est fait via une heuristique. Cette *heuristique de branchement* a fait l'objet de nombreuses études ([Gol79], [JW90], [Fre95], [DABC96], [LA97], [GN02], [DD04], etc.). La variable ainsi choisie est appelée *variable (ou littéral) de décision*.

La procédure DPLL est décrite dans l'algorithme 2.1. Les algorithmes de propagation unitaire et de littéraux purs sont décrits dans les algorithmes 1.1 et 1.2 pages 7 et 8. La procédure `DecisionHeuristic` diffère suivant les implantations de DPLL. Cette fonction est discutée dans la prochaine section.

Heuristiques de branchement

Le choix de la prochaine variable à affecter constitue un facteur déterminant pour l'efficacité de la procédure DPLL. Un arbre de recherche peut être exponentiellement plus « grand » (en nombre de nœuds) en fonction de l'ordre des variables affectées. Cependant, sélectionner la variable optimale à brancher à un certain niveau de l'arbre est un problème NP-difficile [Lib06].

Les heuristiques de branchement cherchent à estimer la variable optimale à affecter. Elles ont fait l'objet de nombreux travaux ces dernières années ([Gol79], [JW90], [Fre95], [DABC96], [LA97], [DD01], [GN02], etc.). Il demeure que toutes ces heuristiques admettent un objectif commun : déterminer la variable qui, par son affectation, va engendrer le plus grand nombre de propagations (unitaires) et fournir ainsi la sous-formule la plus simple à résoudre. Nous illustrons nos propos avec quatre heuristiques qui

sont ou ont été à l'origine de nombreuses autres et demeurent parmi les plus utilisées.

La première est l'heuristique nommée MOMS pour « *Maximum Occurrences in clauses of Minimum Size* » [Gol79]. Comme son nom l'indique, l'idée est simplement de choisir la variable qui apparaît le plus souvent dans les clauses les plus courtes. En effet, leur affectation va raccourcir un maximum de clauses et on peut espérer ainsi favoriser l'application de la règle de propagation unitaire. Cette heuristique a été améliorée à plusieurs reprises, essayant par exemple d'équilibrer les arbres produits par les affectations [JW90, DABC96, DB96].

Le principal inconvénient de MOMS est que l'approximation ne porte que sur les clauses unitaires engendrées directement lors de l'affectation, celles produites par cascade des propagations ne sont pas prises en compte. Pour pallier cet inconvénient, il a été proposé d'utiliser directement la propagation unitaire comme unité de mesure [Fre95]. Un sous-ensemble de variables $\{v_1, v_2, ..., v_n\}$ est choisi. Pour chacune de ces variables, $\Sigma|^*_{v_i}$ et $\Sigma|^*_{\neg v_i}$ sont calculés, la variable choisie est celle pour laquelle $|\mathcal{V}_{\Sigma|^*_{v_i}}| + |\mathcal{V}_{\Sigma|^*_{\neg v_i}}|$ est minimal. Cette heuristique est appelée UP pour « *Unit Propagation* » dans la littérature. Le choix du sous-ensemble de variables à tester peut être fait suivant l'heuristique MOMS mais d'autres critères de sélection se sont avérés plus efficaces [LA97].

Une troisième heuristique se base sur la notion de *backbone* [DD01]. Un littéral appartient au backbone d'une formule si et seulement si il appartient à tous les modèles de la formule. Cette heuristique tente de choisir une variable du backbone du sous-ensemble de clauses courant. Elle a un très bon comportement sur les instances du modèle de génération standard kSAT aléatoire[12].

La dernière heuristique que nous évoquons est actuellement la plus utilisée dans les implantations modernes de DPLL. Cette heuristique estime l'implication de chacune des variables dans les conflits rencontrés, c'est-à-dire les branches contenant la clause vide. Cette heuristique connue sous le nom de VSIDS (« *Variable State Independent Decaying Sum* ») [MMZ+01] associe un compteur à chaque variable. Ce compteur est appelé *activité* de la variable. Lorsqu'un conflit survient, une analyse de conflits (voir § Apprentissage et backjumping) permet de déterminer la raison du conflit et les variables en cause dans ce conflit. Ces variables voient alors leur activité augmenter. L'heuristique consiste à choisir la variable ayant la plus grande activité. Cette heuristique est particulièrement adaptée aux implantations utilisant des structures de données dites paresseuses (voir § Les structures de données paresseuses) pour lesquelles les heuristiques de type MOMS sont impossible à utiliser. À notre connaissance, tous les solveurs modernes de type CDCL (« *Conflict Driven Clause Learning* ») utilisent cette heuristique ou une variante. Le problème de cette heuristique est que les choix faits en haut de l'arbre, c'est-à-dire lorsqu'aucun conflit n'a été rencontré, sont complètement aléatoires. Or ces choix sont certainement les plus importants et influent considérablement sur la taille de l'arbre de recherche. C'est en partie pour ces raisons que les solveurs actuels utilisent une politique de *redémarrage* que nous présentons dans le prochain paragraphe. Pour être complet sur VSIDS, il faut signaler que les activités des variables sont divisées périodiquement par une constante afin « d'oublier » des zones de conflits rencontrés dans d'autres parties de l'arbre de recherche. Enfin, il est à noter que cette notion d'activité de variables est antérieure à VSIDS [BSG98, BGS99].

[12] Ce modèle de génération a été longuement étudié [CS88] et exhibe un phénomène de seuil en dessous duquel toutes les instances générées sont satisfiables et au dessus duquel toutes les instances générées n'admettent plus de modèle. Les instances générées au seuil sont particulièrement difficiles à résoudre. Ce seuil est estimé en pratique à $4{,}25$ pour les instances 3SAT aléatoires. Une instance kSAT est une instance pour laquelle toutes les clauses sont de taille k. La valeur théorique du seuil fait également l'objet de nombreux travaux [DBM00].

Les redémarrages

Les *redémarrages* (« *restarts* ») ont été introduits dans [GSK98] avec l'idée que si la recherche échoue depuis un certain nombre de temps, que l'on évalue en nombre de retours-arrière[13] (« *backtracks* »), alors il est peu probable que la recherche aboutisse en un temps raisonnable et ceci certainement car des mauvais choix ont été réalisés au début de la recherche. Afin de remettre en question ces choix plus rapidement, l'algorithme est relancé avec la formule initiale tout en conservant certaines valeurs (les scores pour VSIDS par exemple) ou les clauses apprises (voir § Apprentissage et backjumping) afin d'effectuer des choix plus judicieux au début de l'arbre.

L'introduction de ces redémarrages peut conduire à perdre la complétude de la méthode. Aussi le nombre de backtracks autorisés augmente d'un redémarrage à l'autre, permettant ainsi de définir différentes politiques de redémarrage. Ces politiques reposent sur différentes séries mathématiques, la plus fréquemment utilisée est certainement la série dite de LUBY [LSZ93] qui est de la forme : 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, ...[14]. Différentes stratégies de redémarrage ont été étudiées dans [Hua07].

Apprentissage et backjumping

Un conflit dans l'algorithme DPLL est la présence de deux littéraux unitaires opposés qui vont conduire inéluctablement à la clause vide par propagation de l'un de ces deux littéraux. Les implantations modernes analysent les raisons de ce conflit afin de déduire une clause dite *assertive* qui est ajoutée à l'ensemble de clauses et qui peut conduire à un retour-arrière non chronologique que l'on appelle *backjumping*. On dit alors que la recherche est guidée par les conflits et par les clauses apprises et on donne alors le nom de CDCL (« *Conflict Driven Clause Learning* ») à l'algorithme.

La clause assertive est obtenue en remontant le *graphe d'implications* à partir des deux littéraux unitaires opposés. Ce graphe acyclique est une représentation des clauses à l'origine des propagations unitaires. Les nœuds de ce graphe sont des littéraux et il existe une arête entre ℓ et les littéraux x_1, x_2, ..., x_n si et seulement si il existe une clause $(\ell \vee \overline{x_1} \vee \overline{x_2} \vee ... \vee \overline{x_n})$ où les x_i sont affectés à un *niveau de décision*[15] inférieur ou égal à celui de ℓ. Partant des deux littéraux opposés dans le graphe d'implication, on calcule initialement la résolvante entre les clauses ayant forcé la propagation de ces littéraux opposés. Puis à partir de cette première résolvante, appelée *clause conflit*, une nouvelle résolvante est calculée en utilisant comme pivot un littéral pour lequel la raison de la propagation est définie, c'est-à-dire pour lequel il existe des prédécesseurs dans le graphe. Les arêtes arrivant sur ce littéral sont toutes étiquetées par la même clause, la résolvante entre la clause conflit et cette clause est calculée et devient la nouvelle clause conflit. Le processus s'arrête lorsque la clause conflit ne contient qu'un seul littéral du niveau de décision courant. La clause conflit est alors appelée clause assertive et le littéral en question le *first UIP* (« *First Unique Implication Point* »).

Soient Σ une CNF et I une interprétation partielle construite sur $\mathcal{V}_\Sigma$ telle que $\Sigma|_I$ contienne la clause vide et que $(\alpha \vee \ell)$ soit la clause assertive calculée à partir de ce conflit avec le littéral $\overline{\ell}$ comme first UIP. Soient $i = \max(\delta(x)|\overline{x} \in \alpha)$, c'est-à-dire le niveau de décision maximum des littéraux de la clause assertive différents du first UIP et $I' \subseteq I$ tel que tous les littéraux de I' aient un niveau de

[13] Un retour-arrière ou « *backtrack* » correspond à l'inversion de la valeur de vérité la variable choisie via l'heuristique de branchement dans DPLL. Cela correspond à la ligne 8 de l'algorithme 2.1.

[14] Formellement, t_i le i^e élément de la série de LUBY se définit comme suit :

$$t_i = \begin{cases} 2^{k-1} & \text{si } \exists k \in \mathbb{N}^* \text{ tel que } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{si } \exists k \in \mathbb{N}^* \text{ tel que } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

[15] À chaque nœud de l'arbre de recherche les littéraux affectés (le littéral de décision et les littéraux propagés) sont étiquetés avec le même niveau de décision. Ce niveau est initialisé à 0 à la racine de l'arbre, incrémenté à chaque nouveau point de décision et décrémenté à chaque backtrack. Pour un littéral ℓ, on note son niveau de décision $\delta(\ell)$.

décision inférieur ou égal à i. Clairement par construction de la clause assertive, nous avons $\Sigma|'_I \vDash^* \{\ell\}$. Ceci permet donc d'effectuer un backjumping jusqu'au niveau de décision i et de considérer la nouvelle interprétation $I' \cup \{\ell\}$.

Ces mécanismes d'ajout de clauses et de retour-arrière non chronologique ont fait et font encore l'objet de nombreux travaux ([MSS96], [BJS97], [ZMMM01], [BKS04], [SBK05], etc.).

Un problème majeur de cette technique concerne le nombre de clauses assertives apprises. Dans l'absolu, l'algorithme est capable d'apprendre une clause pour chaque conflit détecté et le nombre de conflits peut être exponentiel en fonction du nombre de variables de l'instance. Plusieurs techniques ont été proposées dans le but de ne conserver qu'un nombre polynomial de clauses apprises. On cherche bien entendu à garder les clauses qui ont les valeurs informatives les plus importantes, c'est-à-dire celles qui réduiront par la suite le plus possible l'espace de recherche. L'une de ces techniques consiste à ne conserver que les clauses apprises qui ont une taille inférieure ou égale à une longueur donnée [MSS96]. Cependant, la technique, la plus utilisée dans les solveurs actuels, repose sur l'utilisation d'un compteur pour chacune des clauses apprises. Celui-ci voit sa valeur augmentée de 1 dès que la clause à laquelle il est associé, produit une propagation unitaire [GN07]. Régulièrement, la base des clauses apprises est purgée des clauses dont la valeur du compteur est petite.

Dans [BKS04] les auteurs ont prouvé que les approches de type CDCL, couplées avec un nombre non limité de redémarrages, peuvent fournir des preuves exponentiellement plus courtes que les approches n'utilisant pas l'apprentissage.

Les structures de données paresseuses

Dans les solveurs basés sur DPLL, plus de 80% du temps de calcul est consommé par la propagation unitaire. Il était donc primordial de disposer d'une structure de données adaptée permettant d'augmenter de manière substantielle l'efficacité de cette procédure. Lintao ZHANG *et al.* [ZMMM01, ZM02], partant du constat qu'il suffit qu'une clause dispose de deux littéraux non affectés pour ne pas être unitaire, proposent de ne conserver uniquement que deux pointeurs vers deux littéraux de chaque clause. Ces littéraux sont appelés *watched literals*. Lorsque l'un de ces littéraux est affecté à faux le pointeur est déplacé vers un autre littéral de la clause qui n'est pas faux. Lorsqu'il n'est pas possible de trouver deux *watched literals* dans une clause, cela signifie que cette clause est unitaire. L'intérêt de cette structure est triple :

1. capturer la propagation unitaire ;
2. réduire le nombre de clauses à modifier lors de l'affectation d'une variable[16] : si x est mise à vrai seules les clauses pour lesquelles le littéral $\neg x$ est un des deux watched literals sont mises à jour ;
3. accélérer le mécanisme de retour-arrière : aucune mise à jour sur les clauses n'est nécessaire lors d'un « *backtrack* » ou « *backjump* ».

Néanmoins, ce type de structure de données réduit considérablement l'ensemble des heuristiques de branchement qu'il est possible d'utiliser. En effet, toutes les heuristiques, telles que MOMS, qui se basent sur le nombre exact d'occurrences des littéraux ne peuvent plus être utilisées. De ce fait, l'utilisation des watched literals va souvent de paire avec l'heuristique VSIDS.

CDCL : un DPLL moderne

La communauté de recherche sur SAT est partagée quant à la filiation des approches CDCL et DPLL. Néanmoins, les deux algorithmes décrivent un arbre binaire de recherche (plusieurs pour un CDCL utilisant les redémarrages), utilisent tous deux la propagation unitaire pour fixer des littéraux, se fondent

[16] D'où le nom de structure de données paresseuse.

ALGORITHME 2.2 : CDCL

```
   input  : Σ une CNF
   output : vrai si Σ est satisfiable, faux sinon
1  begin
2      I ⟵ ∅ ;                                  /* ensemble des littéraux de décision */
3      nb⊥ ⟵ 0 ;                                /* nombre de conflits rencontrés */
4      while (true) do
5          UnitPropagation (Σ|I) ;
6          if (⊥ ∈ Σ|*I) then                   /* un conflit est détecté */
7              if (I = ∅) then return faux ;    /* clause vide produite à la racine */
8              nb⊥ ⟵ nb⊥ + 1 ;
9              γ ⟵ ConflictAnalysis(Σ,I); /* calcul de la clause assertive */
10             Σ ⟵ Σ ∪ {γ};                     /* ajout de la clause assertive */
11             if (RestartPolicy(nb⊥)) then
12                 bjl ⟵ 0 ; /* redémarrage : backjump à la racine de l'arbre */
13             else
                   /*niveau de backjump calculé en fonction de γ. |I| égal au*/
14                 bjl ⟵ max{δ(ℓ)|ℓ̄ ∈ γ et δ(ℓ) ≠ |I|}; /*niveau de décision courant*/
15             end
16             I ⟵ {ℓ ∈ I|δ(ℓ) ⩽ bjl};          /* backjump au niveau bjl */
17         else if Σ|*I = ∅ then return vrai ;   /* modèle trouvé */
18         else
19             v ⟵ DecisionHeuristic(Σ); /* une décision doit être prise */
20             I ⟵ I ∪ {v} ;        /* passage au niveau de décision supérieur */
21         end
22     end
23 end
```

tous deux sur le principe de séparation et réalisent tous deux des retours-arrière (chronologiques pour l'un, pas forcément pour l'autre). Pour notre part, cela suffit à affirmer que les approches dites CDCL sont les dignes héritières de l'antique procédure de DPLL. Les détracteurs de cette affirmation se contenteront uniquement de comparer les algorithmes 2.1 et 2.2 pour n'y trouver aucune ressemblance !

2.1.2 La recherche locale pour SAT

Les techniques à base de *recherche locale* ont été développées initialement afin de résoudre des problèmes d'optimisation combinatoire. Elles font partie de la famille des métaheuristiques et sont issues de la recherche opérationnelle. Leur application à la recherche d'un modèle pour SAT s'est faite dans les années 90 [SLM92, Gu92, Mor93, JSD96]. Ces méthodes sont, le plus souvent, incomplètes dans le sens où elles sont limitées à la recherche d'un modèle sans garantir son obtention. Elles sont donc, sauf adaptation particulière, incapables de démontrer qu'une formule n'admet pas de modèle. En effet, elles s'appuient sur un parcours non systématique et souvent stochastique de l'espace de recherche.

Le principe de base des méthodes de recherche locale consiste à se déplacer judicieusement dans l'espace des configurations en améliorant la configuration courante. Dans le cas de SAT, les configurations sont des interprétations complètes de la CNF et l'amélioration se juge en terme de nombre de clauses falsifiées par les interprétations. Grossièrement, étant donnée une interprétation, un algorithme

de recherche locale va explorer un *voisinage* de cette interprétation afin d'en découvrir une interprétation satisfaisant plus de clauses que l'interprétation courante. Si une telle interprétation existe, elle est choisie comme nouvelle interprétation courante. Dans le cas contraire, on se trouve dans une situation qualifiée de *minimum local* et une *stratégie d'échappement* est utilisée pour déterminer la nouvelle interprétation courante.

Les méthodes de recherche locale diffèrent essentiellement par le voisinage et la stratégie d'échappement qu'elles utilisent. Généralement le voisinage utilisé est réduit aux interprétations qui diffèrent de l'interprétation courante par un seul littéral, c'est-à-dire les interprétations se trouvant à une distance de HAMMING de 1 de l'interprétation courante. C'est le cas par exemple de l'algorithme GSAT présenté dans [SLM92]. L'algorithme WSAT [SKC93, SKC94] réduit encore ce voisinage aux interprétations distantes de 1 pour lesquelles le littéral qui diffère de l'interprétation courante appartient à une clause donnée. Il existe de nombreuses stratégies d'échappement. Parmi elles, citons : *random walk* [SKC93], *break-out method* [Mor93], *novelty* [MSK97] et variantes (*novelty+* [Hoo99], *novelty++* [LH05]), *survey propagation* [MPZ02, MZ02], *rsaps* [HTH02], *adaptative noise* [Hoo02], *g*^{2}*wsat* [LH05] et ses variantes (*adaptg*^{2}*wsat* [LWZ07]), etc.

En 1997, en collaboration avec Éric GRÉGOIRE et Lakhdar SAÏS nous avons proposé un algorithme de recherche locale reposant sur GSAT et adaptant la méthode tabou [Glo89, Glo90] au cadre SAT **[MSG97c]**. Dans cet algorithme, appelé TSAT[17] pour « *Tabu search for SAT* », le tabou, c'est-à-dire l'interdiction de choisir telle ou telle configuration, ne porte pas sur les interprétations mais sur les variables. Le réglage de la longueur de la liste taboue, c'est-à-dire la durée durant laquelle une configuration est interdite, demeure un paramètre fondamental des méthodes taboues. Nous montrons expérimentalement que dans l'algorithme TSAT, la longueur optimale de cette liste est une fonction linéaire du nombre de variables pour les instances kSAT aléatoires au seuil. Une des motivations qui a conduit à l'élaboration de ce travail est l'élaboration d'une technique atténuant le caractère aléatoire des méthodes de recherche locale, dans le but de mieux appréhender leur comportement dans les situations d'échec comme de succès.

La plupart des algorithmes de recherche locale reposent sur le modèle algorithmique de WSAT. Celui-ci peut-être décrit de la manière suivante, étant donné un nombre maximum de mouvements égal à $\#flips \times \#tries$ où $\#flips$ et $\#tries$ représentent respectivement le nombre de modifications autorisées depuis une configuration initiale et le nombre de configurations initiales testées :

- choisir (le plus souvent aléatoirement) une interprétation initiale I ;
- choisir (le plus souvent aléatoirement) une clause α falsifiée par I ;
- choisir la nouvelle interprétation I' telle que $I \setminus I' = \{\ell\}$ avec $l \in \alpha$;
- remplacer I par I' ;
- répéter les trois derniers points tant que I n'est pas un modèle ou que le nombre de réparations autorisées n'est pas atteint auquel cas l'algorithme peut recommencer depuis la première étape jusqu'à $\#tries$ fois.

Cet algorithme est décrit dans l'algorithme 2.3. L'opération d'inversion de valeur d'une variable qui permet de passer de l'interprétation courante à une de ses voisines, est appelée *flip*.

2.1.3 Hybridation d'algorithmes pour SAT

Depuis que l'hybridation d'algorithmes systématiques et stochastiques pour SAT a été élevée au titre de défi pour la communauté (voir challenge 7 page 14) de nombreux schémas de combinaison ont vu le jour. Citons par exemple la résolution alternée [ZZ96], *walksatz* [HLDV02], *complete local search (cls)* [FR04], *UnitWalk* [HK05], *hbisat* [FH07, FH08], *gunsat* [AS07b], *hinotos* [LMS08], *hybridGM*

[17] La première version de cet algorithme fut présentée en 1995 sous l'appellation « *twsat : tabu walk for SAT* » **[MSG95]**.

ALGORITHME 2.3 : WSAT

input : 2 entiers : $\#flips$ et $\#tries$ et une CNF : Σ
output : vrai si un modèle a été trouvé, faux sinon

```
1  begin
2      for i = 1 to #tries do
3          I ⟵ SelectInitialInterpretation(Σ);
4          for j = 1 to #flips do
5              if (I ⊨ Σ) then return vrai ;
6              α ⟵ ChooseClause(Σ,I) ;
7              ℓ ⟵ RepairStrategy(Σ,I,α) ;
8              I ⟵ (I \ {ℓ}) ∪ {ℓ̄} ;                    /* flip */
9          end
10         if (I ⊨ Σ) then return vrai ;
11     end
12     return faux
13 end
```

[BHG09], etc.

Il est possible d'essayer d'établir une classification des hybridations suivant trois catégories.

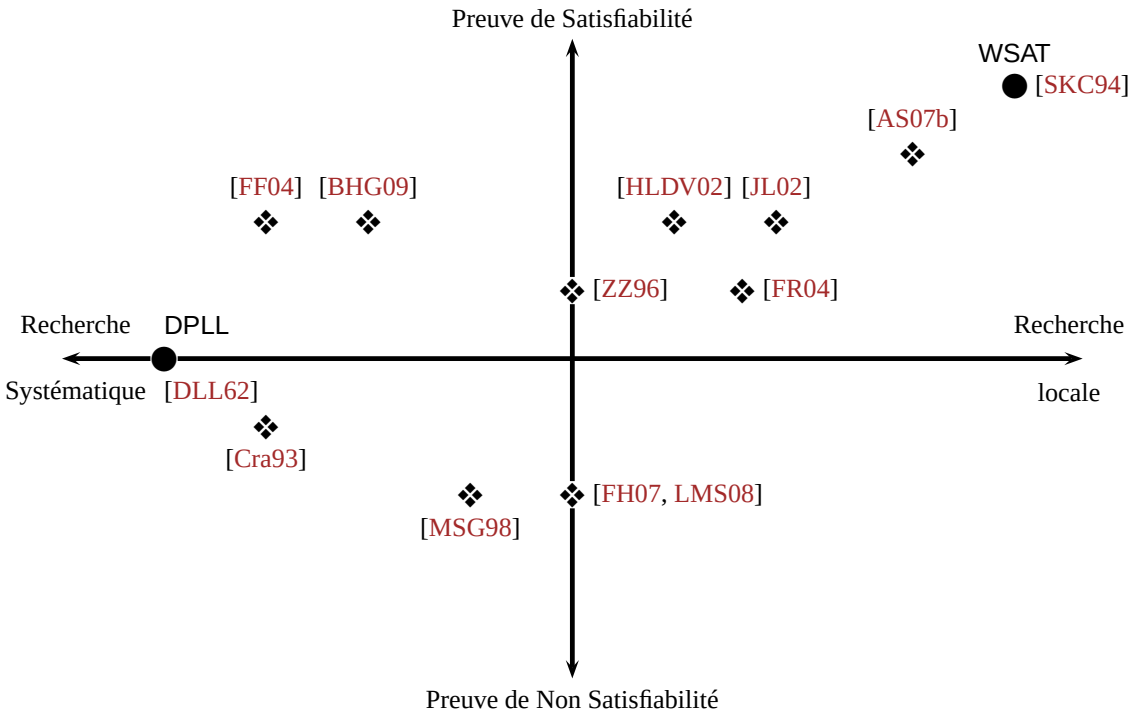
1. La première catégorie regroupe les solveurs pour lesquels DPLL constitue la base du solveur. Dans ce cas, la recherche locale est utilisée la plupart du temps pour guider DPLL [Cra93, MSG98, FF04, HD04]. Toutes ces approches utilisent la recherche locale pour calculer le prochain point de choix. Certaines de ces approches tentent de concentrer la recherche sur des parties non satisfiables de l'instance [MSG98], d'autres sur la partie satisfiable [BHG09, FF04]. De plus, cette étape peut être réalisée avant la recherche [Cra93], à chaque noeud de l'arbre de recherche [MSG98] ou dynamiquement suivant certains critères [FGMS05].

2. La seconde catégorie d'hybridation consiste à utiliser une méthode complète pour venir en aide à la recherche locale [HLDV02, JL02]. Dans [HLDV02], DPLL est utilisé pour trouver des dépendances entre les variables. La recherche locale est ensuite appelée sur un sous-ensemble de variables. Tandis que dans [JL02] (notons que cette méthode est utilisée dans le cadre des CSP), l'approche complète est utilisée pour effectuer un filtrage sur une interprétation partielle. Si celle-ci n'admet pas de solution, une autre interprétation partielle est calculée à l'aide de la recherche locale. Les méthodes proposées dans [FR04, AS07b] utilisent la résolution au sein de la recherche locale pour prouver qu'une instance n'admet pas de modèle.

3. Finalement, la dernière catégorie regroupe les méthodes où les deux approches sont utilisées indépendamment [FH07, LMS08]. La méthode proposée dans [LMS08] est une amélioration de celle proposée dans [FH07]. Dans ces approches, la recherche locale est utilisée pour trouver une solution. Après un temps donné, si aucune solution n'est trouvée, un solveur CDCL est exécuté sur l'ensemble des clauses insatisfaites par l'interprétation courante de la recherche locale. S'il est prouvé que la sous-formule n'admet pas de modèle, le problème est aussi prouvé comme insatisfiable. Ces méthodes sont des variantes modernes de l'algorithme de résolution alternée proposé dans [ZZ96].

La figure 2.1 est une illustration de cette tentative de classification. L'axe des abscisses représente le type de recherche sur laquelle la méthode repose. Il s'étend de DPLL le plus à gauche jusqu'à WSAT le plus à droite. L'axe des ordonnées correspond à la capacité des méthodes à résoudre les instances suivant

FIGURE 2.1 : Tentative de classification de quelques méthodes hybrides



la satisfiabilité de celles-ci. WSAT sur ce graphique se trouve au point le plus haut, étant incapable de résoudre des instances non satisfiables. Cette classification reste extrêmement subjective et peut prêter à discussions. Elle demeure dans ce mémoire uniquement dans le but de permettre aux lecteurs de mieux appréhender ces méthodes.

Par ailleurs, les schémas reposant sur un modèle d'algorithme de recherche locale auquel est greffé un algorithme systématique afin de pouvoir prouver la non satisfiabilité de la CNF testée, apportent des réponses à un second challenge proposé à la communauté dans [SKM97] :

« ***CHALLENGE 5 :*** *Design a practical stochastic local search procedure for proving unsatisfiability.* »

Enfin, il est à noter que l'introduction des redémarrages (voir § Les redémarrages) dans les approches DPLL est également parfois considérée comme un premier pas vers un schéma de combinaison entre méthodes systématiques et stochastiques. De notre point de vue, nous pensons que c'est aller un peu vite en besogne et proposons des schémas où l'imbrication des méthodes est forte permettant à chacune d'entre elles d'exploiter des informations de l'autre.

Dans **[MSG98]**, en collaboration avec Éric GRÉGOIRE et Lakhdar SAÏS, nous avons proposé un schéma d'hybridation où une méthode de recherche locale est appelée à chaque nœud de décision de l'arbre de DPLL. La recherche locale essaye de prolonger l'interprétation partielle construite par DPLL vers un modèle. Autrement dit, les variables qui sont affectées par décision ou propagation lors de DPLL ne sont pas considérées dans la stratégie de réparation de la méthode de recherche locale. Lorsque la méthode de recherche locale échoue, elle retourne à DPLL sa prochaine variable de décision. Cette variable est celle qui est apparue le plus souvent dans les clauses falsifiées durant la recherche locale. Un poids est associé à chaque clause, dès qu'elle est falsifiée son poids augmente de 1. Dès lors, le score d'une variable est égal à la somme des poids des clauses où elle apparaît. Cette stratégie d'associer un poids à chaque clause et d'en augmenter son importance dès lors que la clause est falsifiée a été introduite

ALGORITHME 2.4 : DP+RL

input : Σ une CNF
output : vrai si Σ est satisfiable, faux sinon

```
1  begin
2      Σ* ⟵ UnitPropagation(Σ);
3      if (⊥ ∈ Σ*) then return faux;
4      Σ⊛ ⟵ PurePropagation(Σ*);
5      if (Σ⊛ = ∅) then return vrai;
6      if (LocalSearch(Σ⊛,W_cla)) then return vrai;      /* W_cla poids des clauses */
7      else
8          foreach v ∈ V_Σ⊛ do w(v) ⟵ Σ_{v∈c} W_cla(c) ;
9          v ⟵ v ∈ V_Σ⊛ tel que w(v) est maximal ;
10         if (DPRL(Σ⊛|_v)) then return vrai;
11         else return DPRL(Σ⊛|_¬v);
12     end
13 end
```

par Paul MORRIS dans [Mor93] afin d'extraire sa méthode de recherche locale des minima locaux. Elle est également très similaire à la récente heuristique VSIDS pour DPLL qui augmente le poids d'une clause dès lors qu'elle est intervenue dans la réfutation. Dans cette combinaison la recherche locale et DPLL s'échangent et exploitent des informations : DPLL obtient de la recherche locale sa prochaine variable décision ou éventuellement un modèle alors que la recherche locale travaille sur la sous-formule obtenue à partir de l'interprétation partielle calculée par DPLL et dans laquelle toutes les propagations ont été réalisées. Cette hybridation, sans pour autant surpasser les performances des meilleurs DPLL et des meilleures recherches locales de l'époque, s'est avérée très performante. En effet, elle a permis de résoudre des instances qui posaient des problèmes aussi bien à DPLL qu'à la recherche locale. Ce modèle algorithmique, illustré dans l'algorithme 2.4, fut l'un des premiers schémas d'hybridation efficace et demeure encore aujourd'hui une référence.

L'un des objets de la thèse d'Olivier FOURDINOY [Fou07] que j'ai co-encadrée avec Éric GRÉGOIRE et Lakhdar SAÏS, fut la poursuite du développement de cet algorithme via deux pistes :

1. réduire le nombre d'appels à la méthode de recherche locale ;
2. améliorer la pertinence des poids associés aux clauses.

Si la méthode de recherche locale fournit souvent une variable de décision qui permet de réduire la taille de l'arbre de recherche développé par DPLL, elle demeure extrêmement coûteuse en temps. Aussi nous avons proposé dans **[FGMS05]**, d'appeler la recherche locale uniquement lorsque cela semblait pertinent. Une fois encore deux critères de pertinence ont été retenus et étudiés : la profondeur dans l'arbre de DPLL et le temps écoulé (ou l'espace de recherche parcouru) depuis le dernier appel à la recherche locale. Concernant les poids des clauses, nous avons montré qu'il était inutile de pondérer les clauses au début de la recherche locale. Cela nous a conduit à proposer différentes stratégies de pondération. L'une de ces stratégies consiste à ne pondérer les clauses que dans les minima locaux. Nous exploitons et raffinons cette stratégie pour extraire des MUS (voir § 3.1).

Jusqu'alors les hybridations que nous avons réalisées, portaient sur l'algorithme classique de DPLL et une méthode de recherche locale. En collaboration avec Jean-Marie LAGNIEZ, Gilles AUDEMARD et Lakhdar SAÏS, nous nous sommes intéressés à intégrer des techniques issues des approches CDCL avec les méthodes de recherche locale. Nous avons exploré deux voies :

ALGORITHME 2.5 : CDLS

input : 2 entiers : $\#flips$ et $\#tries$ et une CNF : Σ
output : vrai si un modèle a été trouvé,
faux s'il est prouvé que l'instance ne peut pas admettre de modèle,
$échoué$ sinon

```
1  begin
2      for i = 1 to #tries do
3          I ⟵ SelectInitialInterpretation(Σ);
4          for j = 1 to #flips do
5              if (I ⊨ Σ) then return vrai ;
6              if (∃ℓ ∈ L_{α∈Σ|I∩α=∅} permettant une descente[18]) then I ⟵ (I \ {ℓ̄}) ∪ {ℓ} ;
7              else/* minimum local atteint */
8                  α ⟵ ChooseClause(Σ,I) ; /* choisit aléatoirement une clause
                                              parmi les clauses falsifiées */
9                  β ⟵ ConflictAnalysisLS(Σ,I,α); /* stratégie de réparation
                                                     basée sur l'analyse de conflits */
10                 if (β = ∅) then return faux ;
11                 Σ ⟵ Σ ∪ {β} ;
12             end
13         end
14         if (I ⊨ Σ) then return vrai ;
15     end
16     return faux
17 end
```

1. adapter le concept d'analyse de conflits à partir d'un graphe d'implications au cadre de la recherche locale et exploiter cette analyse pour s'extraire des minima locaux ;
2. utiliser une approche CDCL pour gérer une liste de variables taboues pour la recherche locale ; cette dernière fournissant au CDCL des clauses susceptibles d'appartenir à un MUS afin de construire au plus vite une réfutation.

Dans **[ALMS09b]**, nous proposons une adaptation du concept d'analyse de conflits introduit dans les approches CDCL. Les conflits sont naturellement représentés par les clauses falsifiées par l'interprétation courante de la recherche locale. En revanche, se pose le problème de construire le graphe d'implications et de calculer une clause assertive sans savoir si tel ou tel littéral est décidé ou propagé ou encore sans connaître son niveau de décision. Pour ce faire, nous construisons une nouvelle interprétation partielle à partir de l'interprétation complète courante issue de la recherche locale. Cette interprétation partielle, appelée *interprétation dérivée*, est obtenue en choisissant un littéral à l'identique de l'interprétation courante et en réalisant les propagations engendrées par ce choix. Un littéral est ainsi choisi jusqu'à ce qu'une propagation entraîne un conflit ou une divergence entre l'interprétation complète et sa dérivée. Dans le premier cas, il est alors possible de réaliser une analyse de conflits à l'identique des approches CDCL. Dans le second cas, lorsqu'il existe un littéral ℓ sur lequel les deux interprétations divergent, il suffit d'ajouter artificiellement une raison à $\neg\ell$[19]. Une fois cet ajout réalisé, il est de nouveau possible

[18] Teste s'il existe une variable telle que l'inversion de sa valeur de vérité dans I permet de diminuer le nombre de clauses falsifiées de Σ.

[19] Ceci se fait en considérant une clause non satisfaite par l'interprétation dérivée que l'on rend unitaire en $\neg\ell$ par l'affectation des autres littéraux de la clause.

ALGORITHME 2.6 : `ConflictAnalysisLS()`

input : Σ une CNF, I une interprétation complète de $\mathcal{V}_\Sigma$, α une clause de Σ telle que $I \cap \alpha = \emptyset$
output : β une clause assertive de Σ, I modifiée afin de satisfaire β

begin
  $\Delta \longleftarrow \emptyset$; `/* ensemble des littéraux en conflit avec` α `*/`
  foreach $\beta \in \Sigma | \beta \cap I = \{\ell\}$ avec $\overline{\ell} \in \alpha$ **do** $\Delta \longleftarrow \Delta \cup \{k | k \in \beta \setminus \{\ell\}\}$;
  $dl \longleftarrow 0$; `/* niveau de décision simulé */`
  $I' \longleftarrow \emptyset$; `/* Interprétation partielle dérivée de` I `*/`
  $I'_p \longleftarrow \emptyset$; `/* Liste des littéraux unitaires propagés par` $\Sigma|^*_{I'}$ `*/`
  while $((\bot \in \Sigma|^*_{I'})$ **and** $((I' \cup I'_p) \subset I))$ **do**
    $\Delta \longleftarrow \Delta \setminus (I' \cup I'_p)$;
    $dl \longleftarrow dl + 1$; `/* on simule une nouvelle décision */`
    $I' \longleftarrow I' \cup \{\ell\}$ tel que $\ell \in \Delta$;
    $I'_p \longleftarrow$ `UnitPropagation(`$\Sigma|_{I'}$`)`[20];
  end
  if $(\bot \notin \Sigma|^*_{I'})$ **then**
    $\beta \longleftarrow$ `ConflictAnalysis(`Σ, I'`)` ; `/* calcul de la clause assertive */`
    $I \longleftarrow I \setminus \{x\} \cup \{\overline{x}\}$ avec x first UIP ;
  else
    $y \longleftarrow y \in \mathcal{L}_\Sigma$ tel que $y \in (I' \cup I'_p)$ et $\overline{y} \in I$; `/* littéral qui diffère entre l'interprétation partielle dérivée et l'interprétation complète */`
    $\gamma \longleftarrow \gamma \in \Sigma$ telle que $(\overline{y} \in \gamma)$ et $\gamma \cap (I' \cup I'_p) = \emptyset$;
    foreach $\ell \in \gamma$ **do**
      if $(\ell \neq \overline{y})$ **and** $(\overline{\ell} \notin (I' \cup I'_p)$ **then** $I' \longleftarrow I' \cup \{\overline{\ell}\}$;
    end
    $\beta \longleftarrow$ `ConflictAnalysis(`Σ, I'`)` ; `/* calcul de la clause assertive */`
    foreach $\ell \in (I' \cup I'_p) \setminus I$ **do** $I \longleftarrow I \setminus \{\overline{\ell}\} \cup \{\ell\}$;
  end
  return β ;
end

d'effectuer une analyse de conflits traditionnelle. Dans les deux cas, la clause assertive calculée lors de l'analyse de conflits est ajoutée à la CNF. Cette analyse, illustrée dans l'algorithme 2.6, est réalisée lorsqu'un minimum local est atteint par la méthode de recherche locale. L'objectif premier de cette méthode hybride est d'utiliser l'analyse de conflits pour s'extraire des minima locaux. Pour ce faire, les littéraux de l'interprétation dérivée qui diffèrent de l'interprétation courante, ainsi qu'un littéral de la clause assertive sont flippés. Il est à noter que cet algorithme, nommé CDLS « *Conflict Driven for Local Search* » (voir algorithme 2.5) conserve le caractère incomplet des méthodes de recherche locale mais est capable de prouver la non-satisfiabilité d'une instance si la clause assertive produite est vide. Il apporte ainsi une réponse au challenge 5. Nous l'avons comparé expérimentalement à différentes approches de recherche locale, hybrides, ou CDCL. Sans égaler les approches CDCL sur les instances industrielles, notre approche se montre plus efficace que tous les autres solveurs sur ce type d'instances. Il permet ainsi d'apporter également un élément de réponse à un autre des dix challenges proposés dans [SKM97].

« ***CHALLENGE 6 :*** *Improve stochastic local search on structured problems by efficiently handling*

[20] La fonction `UnitPropagation` (voir algorithme 1.1) est modifiée pour retourner la liste des littéraux qu'elle a propagés.

ALGORITHME 2.7 : SatHyS

Input : Σ une formule CNF et un entier $\#flips$
Output : vrai si Σ est satisfiable, faux sinon

```
1  begin
2    while (true ) do
3      I_c ⟵ SelectInitialInterpretation(Σ) ;
4      I_p ⟵ ∅ ;
5      I_u ⟵ ∅ ;
6      for j = 1 to #flips do
7        if (Σ|_{I_c} = ∅) then return vrai ;
8        if (∃ℓ ∈ L_{α∈Σ|I_c∩α=∅} permettant une descente) and (ℓ̄ ∉ (I_p ∪ I_u)) then
9          I_c ⟵ (I_c \ {ℓ̄}) ∪ {ℓ} ;
10       else
           /*            Minimum local : appel au CDCL            */
11         α ⟵ ChooseClause(Σ,I) ; /* Clause choisie parmi les clauses
                fausses de plus grand degré d'appartenance à un MUS  */
12         I_p ⟵ I_p ∪ {x} tel que x ∈ α ;          /* satisfaction de α */
13         I_c ⟵ (I_c \ {x̄}) ∪ {x} ;                    /* flip de x */
14         I_u ⟵ UnitPropagation (Σ|_{I_p})[21];  /* calcul des propagations */
15         if (⊥ ∈ Σ|*_{I_p}) then
16           γ ⟵ ConflictAnalysis(Σ,I_p) ;
17           if (γ = ⊥) then return faux ;
18           Σ ⟵ Σ ∪ {γ} ;          /* ajout de la clause assertive */
19           bjl ⟵ max{δ(ℓ)|ℓ̄ ∈ γ et δ(ℓ) ≠ |I_p|} ;  /* niveau du backjump */
20           I_p ⟵ {ℓ ∈ I_p|δ(ℓ) ⩽ bjl} ;        /* backjump au niveau bjl */
21           I_u ⟵ {ℓ ∈ I_u|δ(ℓ) ⩽ bjl} ;
22         end
23       end
24     end
25   end
26 end
```

variable dependencies. »

L'approche CDLS, bien que capable de montrer qu'une instance n'admet pas de modèle, reste incomplète. Dans **[ALMS09a]**, nous avons proposé un modèle algorithmique hybride combinant approche CDCL et recherche locale qui demeure complet bien que reposant principalement sur une méthode de recherche locale. Le processus de recherche consiste à se déplacer d'une interprétation à une interprétation voisine jusqu'à obtenir une solution ou jusqu'à ce que la preuve de l'absence de modèle soit faite. À chaque étape, on essaie de réduire le nombre de clauses falsifiées (phase de descente). Lorsqu'un minimum local est atteint une approche de type CDCL est appelée. Cette approche sélectionne comme littéral de décision un littéral appartenant à une clause falsifiée par l'interprétation courante de la recherche locale. La clause est choisie en fonction de son degré[22] présumé d'appartenance à un MUS de l'instance.

[21] Comme précédemment, la fonction `UnitPropagation` (voir algorithme 1.1) est modifiée pour retourner la liste des littéraux qu'elle a propagés.

[22] Lorsqu'une clause est falsifiée par une interprétation courante, si on suppose que l'instance ne possède qu'un seul MUS, son degré d'appartenance est égal à $\frac{1}{n}$ où n est le nombre de clauses falsifiées par l'interprétation courante. Chaque fois qu'une

TABLE 2.1 : Nombre d'instances résolues comparativement à SatHyS

		Crafted		Industrial		Random	
		sat	¬sat	sat	¬sat	sat	¬sat
adaptg²wsat	[LWZ07]	326	0	232	0	*1111*	0
rsaps	[HTH02]	339	0	226	0	1071	0
WSAT (*random walk*)	[SKC94]	259	0	206	0	1012	0
cls	[FR04]	235	75	227	102	690	0
hybridGM	[BHG09]	290	0	209	0	**1114**	0
SatHyS	[ALMS09a]	*340*	*174*	*473*	*266*	930	*17*
minisat	[ES04]	**402**	**369**	**588**	**414**	609	**315**

L'approche CDCL fixe ce littéral de manière à satisfaire la clause et effectue la propagation unitaire. L'ensemble des littéraux fixés par le CDCL devient tabou pour la méthode de la recherche locale, c'est-à-dire qu'elle ne peut plus flipper les variables de cet ensemble. Si la décision prise par l'approche CDCL conduit à un conflit, une clause assertive est ajoutée à la CNF. Si le niveau du conflit est nul, l'instance est prouvée ne pas admettre de modèle. L'approche CDCL fournit donc à la recherche locale une liste de variables taboues et concentre sa recherche sur la preuve de la non-satisfiabilité de l'instance en privilégiant des clauses que la recherche locale a jugées susceptibles d'appartenir à un MUS. Cet algorithme, nommé SatHyS, a été comparé à différentes approches de recherche locale, hybrides et CDCL. Bien que son développement soit récent et loin d'être mature, les résultats obtenus par cet algorithme sont particulièrement remarquables. En effet, si les performances des approches CDCL pur ne sont pas encore atteintes, elles sont approchées et celles des autres méthodes (de recherche locale et hybrides) sont dépassées. Un extrait des résultats obtenus sur l'ensemble des instances soumises aux dernières compétitions SAT est fourni dans la table 2.1. On voit clairement que SatHyS obtient des résultats très honorables sur les catégories d'instances industrielles, là où les autres approches reposant sur une recherche locale sont à la peine.

2.2 Exploitation des propriétés structurelles

Une autre voie pour étendre la classe des problèmes traitables en pratique est d'exploiter la nature des problèmes à résoudre. Plus précisément, l'objectif est de détecter et d'exploiter les propriétés structurelles intrinsèques du problème qui sont souvent masquées par le formalisme propositionnel. Parmi ces propriétés structurelles, nous pouvons citer les *symétries* [BS94], les *backdoors* [WGS03], les *classes polynomiales* [Héb08], les *autarkies* [MS85, VG99, Kul00], les redondances [Lib05, Lib08a, Lib08b, BR00], etc.

Nous avons travaillé sur quatre aspects ayant trait aux propriétés structurelles :

– les dépendances fonctionnelles ;
– les redondances ;
– les classes polynomiales ;
– les autarkies.

2.2.1 Les dépendances fonctionnelles

La représentation sous forme normale conjonctive de problèmes issus d'applications réelles ne permet pas une représentation simple et concise des connaissances ; alors qu'à l'inverse la logique proposi-

clause apparaît falsifiée dans un minimum local, son degré d'appartenance à un MUC est augmenté de $\frac{1}{n}$.

tionnelle, lorsqu'elle est utilisée dans sa totalité, permet de représenter simplement une grande variété de connaissances tout en préservant la structure de la connaissance. Les informations structurelles souvent perdues lors de la transformation en CNF peuvent s'avérer utiles pour la résolution [KMS97, RSB99].

Nous nous sommes intéressés à la détection et à l'exploitation de *portes booléennes* dans une CNF. Une porte (booléenne) est une expression de la forme $y = f(x_1, x_2, ..., x_n)$ où f est un opérateur logique parmi $\{\vee, \wedge, \Leftrightarrow\}$ et où y et x_i sont des littéraux propositionnels. Ces portes que l'on rencontre fréquemment dans des instances SAT codant des problèmes industriels comme la vérification de circuits, se traduisent sous forme CNF de la manière suivante :

- $y = \wedge(x_1, x_2, ..., x_n)$ peut être représenté par l'ensemble de clauses : $\begin{cases} (y \vee \overline{x_1} \vee \overline{x_2} \vee ... \vee \overline{x_n}) \\ (\neg y \vee x_1) \\ (\neg y \vee x_2) \\ ... \\ (\neg y \vee x_n) \end{cases}$,
  traduisant le fait que la valeur de y est entièrement déterminée par les valeurs de x_i s.t. $i \in [1..n]$;
- $y = \vee(x_1, x_2, ..., x_n)$ représente l'ensemble de clauses $\begin{cases} (\neg y \vee x_1 \vee ... \vee x_n) \\ (y \vee \overline{x_1}) \\ (y \vee \overline{x_2}) \\ ... \\ (y \vee \overline{x_n} \end{cases}$;
- $y = \Leftrightarrow (x_1, x_2, \ldots, x_n)$ représente la *chaîne d'équivalences* (aussi appelée *formule biconditionnelle*) $y \Leftrightarrow x_1 \Leftrightarrow ... \Leftrightarrow x_n$, qui est équivalente à 2^n clauses.

Par la suite, nous considérons les portes de la forme $y = f(x_1, x_2, ..., x_n)$ où y est une variable ou la constante booléenne vrai. En effet, une clause peut être représentée par une porte de la forme $vrai = \vee(x_1, x_2, ..., x_n)$. De plus, une porte $\neg y = \wedge(x_1, x_2, ..., x_n)$ (respectivement $\neg y = \vee(x_1, x_2, ..., x_n)$) est équivalente à $y = \vee(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$ (respectivement $y = \wedge(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$). À partir de la propriété sur les chaînes d'équivalences, énonçant que, dans une chaîne d'équivalences, les négations peuvent être supprimées par paires, il est possible de réécrire toute chaîne d'équivalences en une chaîne équivalente contenant au plus une négation. Ceci permet de réécrire chaque porte de la forme $y = \Leftrightarrow (x_1, x_2, ..., x_n)$ en une porte où y est un littéral positif. Par exemple, la porte $\neg y = \Leftrightarrow (\neg x_1, x_2, x_3)$ est équivalente à $y = \Leftrightarrow (x_1, x_2, x_3)$ et $\neg y = \Leftrightarrow (\neg x_1, x_2, \neg x_3)$ est équivalente par exemple à $y = \Leftrightarrow (x_1, x_2, \neg x_3)$.

Une variable propositionnelle y (respectivement $x_1, \ldots, x_n$) est une *variable de sortie* (respectivement sont *des variables d'entrée*) d'une porte de la forme $y = f(x'_1, x'_2, ..., x'_n)$, où $x'_i \in \{x_i, \neg x_i\}$.

Une variable propositionnelle z est une *variable de sortie (dépendante) d'un ensemble de portes* si et seulement si z est une variable de sortie d'au moins une porte de cet ensemble. Une *variable d'entrée (indépendante) d'un ensemble de portes* est une variable qui n'est variable de sortie d'aucune porte de cet ensemble.

Une porte est satisfaite sous une interprétation si et seulement si les parties gauche et droite de la porte sont toutes les deux à vrai ou à faux sous cette interprétation. Une interprétation satisfait un ensemble de portes si et seulement si chaque porte de cet ensemble est satisfaite sous cette interprétation. Une telle interprétation est appelée modèle de cet ensemble de portes.

À notre connaissance, jusqu'alors seules les chaînes d'équivalences ont été étudiées et utilisées au sein d'un solveur de type DPLL : *eqsatz* [Li00]. Cette adaptation de *satz* [LA97], fait une recherche syntaxique de chaînes d'équivalences de taille deux ou trois et grâce à cette détection, fut l'une des premières méthodes à résoudre les instances *parity-32* proposées par Joost WARNERS et Hans VAN MAAREN [WvM99] qui représentaient un challenge pour la communauté [SKM97] :

« ***CHALLENGE 2 :*** *Develop an algorithm that finds a model for the DIMACS 32-bit parity problem.* »

Dans l'objectif d'apporter une nouvelle solution à ce challenge, nous nous sommes intéressés à la

détection des portes $\wedge$, $\vee$ et de chaînes d'équivalences de longueur quelconque. En collaboration avec Éric GRÉGOIRE, Richard OSTROWSKI et Lakhdar SAÏS, nous avons exhibé, dans **[OGMS02]**, une propriété concernant ces portes permettant de limiter le nombre de tests syntaxiques nécessaires à cette reconnaissance.

Propriété 1. Toutes les clauses produites par résolution entre deux clauses issues de la représentation clausale d'une même porte sont tautologiques.

Partant de cette propriété, nous avons proposé un algorithme en deux étapes pour détecter les portes dans une CNF :

1. isoler un sous-ensemble de clauses susceptibles d'appartenir à la même porte, c'est-à-dire pour lesquelles les résolvantes sont tautologiques ;
2. et identifier syntaxiquement la porte à partir de ce sous-ensemble.

Une fois les portes identifiées, les clauses représentant des portes sont remplacées par les portes correspondantes afin d'obtenir une formule hybride. Un DPLL adapté à la résolution de cette formule hybride a été développé et s'est montré extrêmement efficace quant à la résolution de certaines instances et particulièrement des instances *parity-32*, résolvant ces instances jusqu'à plus de 130 fois plus rapidement que le solveur *eqsatz* (temps de prétraitement inclus). La formulation hybride obtenue permet à la fois d'exploiter les propriétés structurelles de la formule initiale mais également de diviser en deux parties l'ensemble des variables de la formule. En effet, les variables apparaissant uniquement en sortie des portes peuvent être exclues des stratégies de décision. Par exemple, toujours sur les instances *parity-32*xs, nous avons pu restreindre l'ensemble des variables de décision à uniquement 32 variables, réduisant ainsi l'espace de recherche à 2^{32} au lieu de 2^{3176} !

Il est parfois délicat de scinder en deux ensembles disjoints l'ensemble des variables d'entrée et de sortie d'une formule. En effet, il peut exister des cycles dans les définitions des variables, par exemple : $\{x = \wedge(y, z), y = \vee(x, \neg t)\}$. Dans ce cas, il faut déterminer un ensemble de variables coupe-cycle qui seront associées aux variables d'entrée. Dans l'exemple précédent x peut jouer ce rôle, l'ensemble des variables d'entrée devenant $\{x, z, t\}$. Trouver l'ensemble minimal de variables coupe-cycle est un problème NP-difficile, des heuristiques devront être employées pour son calcul. Nous avons proposé et comparé différentes heuristiques dans **[GMOS05]**. Il est à noter que cet ensemble de variables d'entrée correspond à un *strong backdoor* comme défini dans [WGS03].

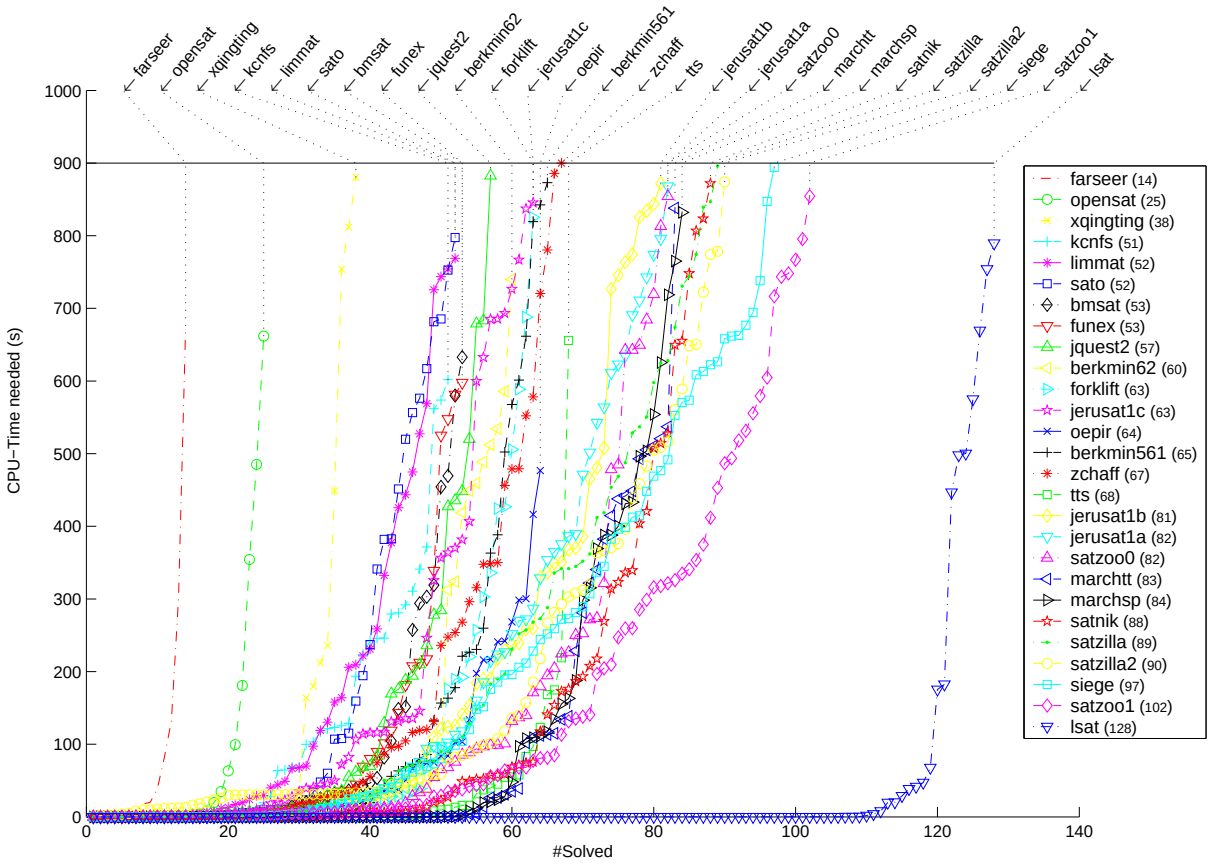
Même si la détection de portes proposée s'avère relativement efficace en pratique, il demeure que l'étape de reconnaissance syntaxique peut être prohibitive sur certaines instances. Dans **[GMOS05]**, nous exploitons la propriété suivante afin de détecter les portes :

Propriété 2. Soient Σ une CNF, $\ell \in \mathcal{L}_\Sigma$ et $\alpha \in \Sigma$ telle que $\alpha = \{\ell, \ell_1, \ell_2, ..., \ell_n\}$. Si $\{\overline{l_1}, \overline{l_2}, ..., \overline{l_n}\} \subseteq UP(\Sigma|_\ell)$[23] alors $\Sigma \vDash (\ell = \wedge(\overline{l_1}, \overline{l_2}, ..., \overline{l_n}))$.

En fonction du signe du littéral ℓ, les portes $\wedge$ et $\vee$ peuvent être détectées. Cette propriété permet également de découvrir des équivalences binaires ($a = \wedge(b)$ étant équivalent à $a \Leftrightarrow b$). Par ailleurs, il est possible de détecter des portes qui ne l'auraient pas été par la méthode syntaxique. Par exemple, l'ensemble de clauses $\{(y \vee \neg x_1 \neg x_2, \neg x_3), (\neg y \vee x_1), (\neg x_1 \vee x_4), (\neg x_4 \vee x_2), (\neg x_2 \vee x_5), (\neg x_4 \vee \neg x_5 \vee x_3)\}$ code la porte $y = \wedge(x_1, x_2, x_3)$ qui n'est pas détectable sous cette représentation avec la méthode syntaxique (la condition nécessaire, mais non suffisante, d'appartenance à une porte n'est pas satisfaite : toutes les résolvantes ne sont pas tautologiques). Une nouvelle technique de recherche de portes consiste donc à vérifier la propriété 2 pour chaque littéral apparaissant de $\mathcal{L}_\Sigma$. Il est à noter que ce traitement

[23] Nous désignons par $UP(\Sigma)$ l'ensemble des littéraux propagés durant le processus de simplification par les littéraux unitaires de Σ.

FIGURE 2.2 : Résultats de LSAT lors de la première phase de la compétition internationale SAT'2003 sur les instances dites « *crafted* ».



s'appuie sur la propagation unitaire qui est un processus linéaire en temps et demeure donc polynomial par rapport à la taille de Σ.

Cette technique de détection a été implantée en tant que prétraitement au sein d'un solveur de type DPLL. Ce solveur, appelé LSAT , a été soumis à la compétition internationale SAT organisée en 2003 (`http://www.satcompetition.org/`). Il a obtenu des résultats très remarqués comme l'illustre la figure 2.2 extraite des résultats officiels de la compétition. Une description complète de LSAT peut être trouvée dans la thèse de Richard OSTROWSKI [Ost04] que j'ai co-encadrée.

Ces travaux sur la détection de dépendances fonctionnelles à l'aide de portes ont été appliqués à la recherche locale dans [PTS07]. Dans la suite de nos travaux concernant l'exploitation de propriétés structurelles, nous avons poursuivi notre idée d'exploiter la propagation unitaire pour simplifier la formule et notamment pour tester la redondance d'une clause dans une CNF.

2.2.2 La u-redondance

Les redondances dans les formules propositionnelles ont fait l'objet de nombreuses études [Lib05, Lib08a, Lib08b]. Concernant la représentation CNF, cette redondance s'exprime essentiellement au niveau des clauses. On dit qu'une clause α est *redondante* pour une CNF Σ si et seulement si $\Sigma \setminus \{\alpha\} \models \alpha$ [BR00]. Clairement, décider si une clause donnée est redondante pour un ensemble de clauses est CoNP-complet. Néanmoins, les clauses redondantes jouent un rôle fondamental dans l'efficacité des solveurs SAT. Ceci est notamment attesté par l'ajout des clauses assertives dans les méthodes CDCL qui sont par

ALGORITHME 2.8 : `U-Irredondancy()`

input : Σ une CNF
output : une CNF u-irredondante minimale en terme de taille obtenue à partir de Σ

```
1 begin
2     foreach α = {ℓ1, ℓ2, ..., ℓn} ∈ Σ triées par ordre décroissant de taille do
3         Σ ⟵ Σ \ {α} ;
4         if (⊥ ∉ UnitPropagation(Σ|{¬ℓ1,¬ℓ2,...,¬ℓn})) then Σ ⟵ Σ ∪ {α} ;
5     end
6     return Σ;
7 end
```

construction redondantes. L'intérêt de ces clauses redondantes n'est donc plus à démontrer. Néanmoins, certaines redondances peuvent également « polluer » l'ensemble de clauses au point de rendre l'instance plus difficile à résoudre. De ce point de vue, nous avons étudié l'impact des redondances modulo la propagation unitaire.

Une clause α est dite *u-redondante* pour une CNF Σ si et seulement si $\alpha \in \Sigma$ et $\Sigma \setminus \{\alpha\} \vDash^* \alpha$. Contrairement au test de redondance qui est exponentiel dans le pire cas, le test d'u-redondance est linéaire en temps. Par extension, une CNF est dite *u-irredondante* si et seulement si $\forall \alpha \in \Sigma$, α n'est pas u-redondant.

En collaboration avec Olivier FOURDRINOY, Éric GRÉGOIRE et Lakhdar SAÏS nous avons étudié l'impact des clauses u-redondantes sur l'efficacité des solveurs modernes. Pour ce faire, nous avons proposé, dans **[FGMS07a]**, un nouveau prétraitement permettant de supprimer de l'ensemble de clauses les clauses u-redondantes. Ce prétraitement, détaillé dans l'algorithme 2.8 délivre une formule u-irredondante. De manière évidente, une formule u-irredondante ne contient aucune clause subsumée. Par ailleurs, nous avons montré également que si le processus de test de suppression des clauses u-redondantes commence par les clauses les plus longues, la formule obtenue est à la fois minimale en terme du nombre de clauses mais également en terme du nombre de littéraux par clause. Les résultats expérimentaux ont montré qu'il était plus souvent bénéfique de supprimer ces clauses que de les conserver, permettant parfois de résoudre certaines instances non résolues sans ces suppressions. Néanmoins, le coût de cette transformation peut s'avérer prohibitif, nous avons donc proposé de ne tester que les clauses non Horn ou non binaires ou satisfaisant un critère heuristique estimant la probabilité d'u-redondance d'une clause.

Ces travaux ont été étendus par Cédric PIETTE [Pie08] qui propose de laisser les solveurs décider de l'utilité des clauses u-redondantes en ne les supprimant pas mais en les plaçant dans la zone des clauses apprises par le solveur. Dans la section suivante, nous utilisons cette u-redondance afin de plonger des instances dans des fragments polynomiaux.

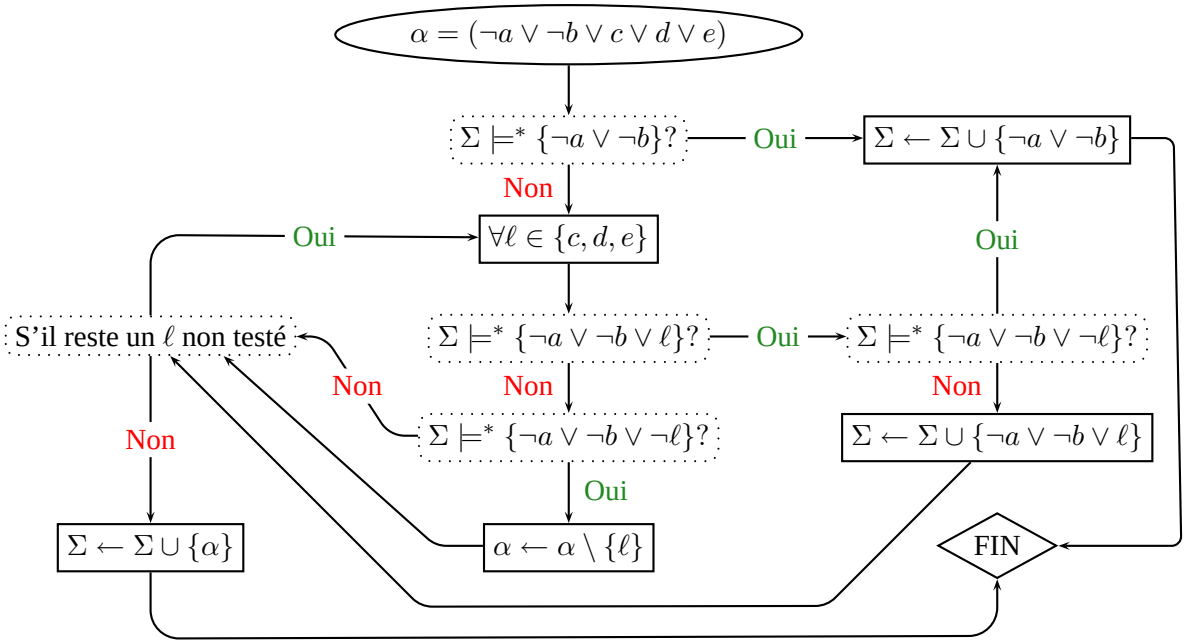
2.2.3 Classes polynomiales modulo la u-redondance

Il existe de nombreuses classes polynomiales[24] pour SAT (HORNSAT [Hor51], 2SAT [Coo71], les instances HORN-renommables [Lew78], les formules bien imbriquées [Knu90], les différentes hiérarchies [YD83, GS88, DE92], les restrictions sur les occurrences [Tov84], les clauses q-HORN [BCHS90], QUAD [Dal96], etc.

Étant donnée une CNF quelconque et une classe polynomiale reposant sur des critères syntaxiques (HORN, *reverse*-HORN, clauses positives, négatives, binaires, etc.) il est possible de diviser l'ensemble

[24] Une classe polynomiale est un ensemble d'instances, pour lequel il existe un algorithme d'appartenance à cet ensemble et un algorithme de résolution qui s'exécutent tous deux en temps polynomial.

FIGURE 2.3 : Traitement d'une clause lors du test d'appartenance à la classe U-HORN



des clauses en deux : le premier contenant les clauses pour lesquelles le critère syntaxique est satisfait ; le second contenant les clauses restantes. Cette dernière partie est appelée le *fragment non polynomial* de l'instance SAT.

L'idée principale est de réduire le fragment non-polynomial d'une instance SAT via la propagation unitaire. Plus précisément, si l'ensemble des clauses du fragment non-polynomial est u-redondant, alors il est possible de le supprimer et de plonger ainsi l'instance dans une classe polynomiale existante. Nous nommons par U-$\mathcal{C}$ les nouvelles classes polynomiales ainsi construites, avec $\mathcal{C}$ une classe polynomiale reposant sur des critères syntaxiques.

Dans **[FGMS07b]**, nous nous sommes particulièrement intéressés à réduire le fragment non-HORN d'une CNF et donc à la définition de la classe polynomiale U-HORN . Lorsqu'une clause du fragment non-polynomial s'avère ne pas être u-redondante, il est tout de même possible de plonger cette clause dans le fragment polynomial en réduisant sa partie positive. Pour ce faire, on teste si chaque sous-clause de HORN constructible à partir de cette clause est une conséquence logique restreinte à la propagation unitaire. Si c'est le cas, la clause initiale peut être remplacée par une sous-clause de HORN. D'autre part, afin de réduire au maximum la partie positive de la clause, on vérifie également que les clauses totalement négatives obtenues à partir des sous-clauses de HORN précédemment testées ne sont pas des conséquences logiques restreintes à la propagation unitaire de la CNF. Si c'est le cas, il est alors possible d'effacer (par simple résolution) le littéral dans la clause initiale, permettant ainsi de réduire la partie positive de la clause. La figure 2.3 synthétise les différents tests subis par une clause non-HORN, tandis que l'algorithme 2.9 fournit un algorithme polynomial de détection d'appartenance à la classe U-HORN[25]. Nous avons exécuté cet algorithme de détection sur 1600 instances issues des dernières compétitions SAT (`http://www.satcompetition.org`), 127 instances se sont révélées être U-HORN, soit près de 8%. Nous pensons qu'il est fort probable que de nombreuses autres instances appartiennent à d'autres

[25]La formule obtenue étant HORN, l'algorithme de résolution est donc la propagation unitaire. Si elle produit une clause vide, l'instance n'admet pas de modèle, sinon elle est satisfiable.

ALGORITHME 2.9 : isU-Horn()

input : Une instance SAT Σ
output : **true** si Σ est U-HORN ; **false**

begin
  $\Sigma' \longleftarrow \{\alpha | \alpha$ est une clause de HORN de $\Sigma\}$;
  foreach $\alpha \in \Sigma$ *telle que* $\alpha = \{\neg n_1, \ldots, \neg n_{n'}, p_1, \ldots, p_{p'}\}$ *avec* $n' \geq 0$ *et* $p' > 1$ **do**
    if $(\Sigma \vDash^* (\neg n_1 \vee \cdots \vee \neg n_{n'}))$ **then** $\Sigma' \longleftarrow \Sigma' \cup \{(\neg n_1 \vee \cdots \vee \neg n_{n'}\}\}$;
    else
      $\Sigma'' \longleftarrow \emptyset$;
      $\alpha' \longleftarrow \alpha$;
      foreach $p_i \in \alpha$ **do**
        if $(\Sigma \vDash^* (\neg n_1 \vee \cdots \vee \neg n_{n'}, p_i))$ **then**
          if $(\Sigma \models^* (\neg n_1 \vee \cdots \vee \neg n_{n'} \vee \neg p_i))$ **then** $\Sigma' \longleftarrow \Sigma' \cup \{(\neg n_1 \vee \cdots \vee \neg n_{n'})\}$;
          else $\Sigma'' \longleftarrow \Sigma'' \cup \{(\neg n_1 \vee \cdots \vee \neg n_{n'} \vee p_i)\}$;
        end
        else if $(\Sigma \models^* (\neg n_1 \vee \cdots \vee \neg n_{n'} \vee \neg p_i))$ **then** $\alpha' \longleftarrow \alpha' \setminus \{p_i\}$;
      end
      if $((\neg n_1 \vee \cdots \vee \neg n_{n'}) \notin \Sigma')$ **then**
        if $(\Sigma'' = \emptyset)$ **then** $\Sigma' \longleftarrow \Sigma' \cup \{\alpha'\}$;
        else $\Sigma' \longleftarrow \Sigma' \cup \Sigma''$;
      end
    end
  end
  $\Sigma' \longleftarrow$ `U-Irredondancy(`Σ'`)` ;
  return `isHorn(`Σ'`)` ;
end

classes polynomiales de type U-$\mathcal{C}$.

L'utilisation de la conséquence logique restreinte à la propagation unitaire comme mécanisme de déduction permet ici de produire un algorithme demeurant polynomial en temps. Nous avons appliqué une idée similaire au concept d'autarky afin de généraliser cette notion tout en conservant ce caractère polynomial.

2.2.4 Les autarkies

La notion d'*autarky* a été introduite dans [MS85]. Une autarky est une interprétation partielle I d'une CNF Σ telle que $\Sigma|_I \subseteq \Sigma$. Un cas particulier d'autarky est l'ensemble des littéraux purs de la formule. Cette notion d'autarky a été utilisée au sein d'un solveur SAT afin d'en accroître son efficacité [JOR88, VG99, Kul00].

Dans **[GMS09]**, en collaboration avec Éric GRÉGOIRE et Lakhdar SAÏS, nous proposons de remplacer le test d'inclusion par un test de conséquence logique restreinte à la propagation unitaire, définissant ainsi le concept d'autarky généralisé modulo la propagation unitaire. Plus précisément, une interprétation partielle I d'une CNF Σ est une autarky généralisée modulo la propagation unitaire si et seulement si il existe $I' \subset I$ telle que $\Sigma|_{I'} \vDash^* \Sigma|_I$. Si $I' \neq \emptyset$, on qualifie cette autarky de locale. Nous exhibons des propriétés concernant ces autarkies permettant d'intégrer efficacement leur détection au sein d'un algorithme de type DPLL ou CDCL. Nous montrons également que la recherche de ces autarkies

peut engendrer un *backjump* plus important que le processus d'analyse de conflits traditionnelle (voir §Apprentissage et backjumping).

2.3 Techniques de simplification

Prétraiter une CNF avant sa résolution est devenu une étape fondamentale pour l'efficacité de sa résolution. Un grand nombre de préprocesseurs ont été proposés ces dernières années. L'un des premiers algorithmes de prétraitement efficace, est la *3-Resolution*, intégré notamment aux solveurs *csat* [DABC96] et *satz* [LA97]. Celui-ci consiste à ajouter à la formule toutes les résolvantes de taille inférieure ou égale à 3, jusqu'à saturation. De plus, les clauses subsumées sont détectées et supprimées de la CNF. Un préprocesseur moins gourmand en ressources a ensuite été proposé dans [Bra01]. Du nom de *2-sim*, il a été développé afin d'améliorer l'efficacité des solveurs sur les problèmes industriels, qui contiennent souvent un grand nombre de clauses binaires. Sommairement, l'idée est donc d'utiliser ces clauses binaires pour construire un graphe d'implications, à partir duquel des clauses unitaires sont dérivées par le calcul de la fermeture transitive. Si de telles clauses sont produites, elles sont propagées et ce procédé est itéré jusqu'à ce qu'un point fixe soit atteint. Le préprocesseur *hypre* a généralisé *2-sim* en utilisant l'hyper-résolution pour déduire de nouvelles clauses binaires [BW04]. De plus, cet algorithme détecte et substitue les littéraux équivalents de manière incrémentale. La procédure DP, basée sur l'élimination de variables par résolution, a également été considérée comme prétraitement d'une CNF. Sa complexité exponentielle en espace a obligé l'adoption d'un schéma plus faible, comme celui proposé dans *niver* [SP05]. Ce schéma consiste à éliminer les variables par résolution uniquement si ce calcul ne conduit pas à augmenter le nombre de littéraux de la CNF. Ce prétraitement a ensuite été amélioré par l'utilisation d'une nouvelle règle de substitution et de *signatures* de clauses [EB05]. Le préprocesseur résultant, nommé *satelite*, est depuis utilisé comme étape préliminaire de la plupart des solveurs actuels.

Notre contribution aux algorithmes de prétraitement a déjà en partie été présentée en section 2.2.1 où un prétraitement des formules CNF permettant d'exhiber des dépendances fonctionnelles sous la forme de portes logiques est décrit. Les prétraitements présentés dans cette partie du document peuvent également être appliqués dynamiquement au cours de la recherche dans un objectif de simplification. Néanmoins, pour des raisons évidentes d'efficacité, il est souvent préférable de les appliquer uniquement sur la formule initiale, c'est-à-dire à la racine de l'arbre de recherche.

2.3.1 Clauses nf-bloquées

Dans la section 2.2.2, nous avons étudié les clauses u-redondantes et montré que leur suppression plongeait de nombreuses instances dans des classes polynomiales (voir §2.2.3). Nous nous sommes intéressés à un autre type de clauses particulières : les *clauses bloquées*.

Le concept de clause bloquée a été introduit par Oliver KULLMANN [Kul96]. Une clause α est dite bloquée pour une CNF Σ si et seulement si il existe un littéral ℓ de α tel que toutes les résolvantes en ℓ de α sont tautologiques. Une telle clause ne peut participer à aucune réfutation, il en découle que Σ est satisfiable si et seulement si $\Sigma \setminus \{\alpha\}$ est satisfiable.

Exemple 8. Soit $\Sigma = \left\{ \begin{array}{l} \alpha_1 : (a \vee b \vee c) \\ \alpha_2 : (\neg a \vee \neg b) \\ \alpha_3 : (\neg b \vee c) \\ \alpha_4 : (b \vee \neg c) \end{array} \right\}$ la clause α_1 est bloquée par le littéral a.

Dans **[OGMS02]**, en collaboration avec Éric GRÉGOIRE, Richard OSTROWSKI et Lakhdar SAÏS nous avons étendu la notion de clause bloquée en celle de *clause nf-bloquée*. Une clause est dite *non*

fondamentale pour une CNF, si elle est tautologique ou si elle est subsumée par une autre clause de la CNF. Une clause α est nf-bloquée pour une CNF Σ, s'il existe un littéral ℓ de α tel qu'il n'existe pas de résolvante en ℓ[26] ou toutes les résolvantes sont non fondamentales. Très clairement, comme pour les bloquées, si α est nf-bloquée dans Σ, alors Σ est satisfiable si et seulement si $\Sigma \setminus \{\alpha\}$ est satisfiable.

Exemple 9. Soit $\Sigma = \left\{ \begin{array}{l} \alpha_1 : (a \vee b \vee c) \\ \alpha_2 : (\neg a \vee b \vee d) \\ \alpha_3 : (b \vee c \vee d) \\ \alpha_4 : (\neg b \vee c \vee \neg d) \\ \alpha_5 : (a \vee b \vee \neg c) \end{array} \right\}$ la clause α_1 est bloquée par le littéral a. α_1 peut donc être supprimée, ce qui permet de nf-bloquer α_2 par le littéral b. Si l'on poursuit les suppressions, la clause α_3 devient nf-bloquée par d et enfin les deux clauses restantes peuvent également être supprimées puisqu'elles possèdent chacun un littéral pur ($\neg d$ pour α_4 et a pour α_5). Nous obtenons l'ensemble vide de clause et avons ainsi prouvé que Σ est satisfiable. Prétraiter les instances afin de supprimer les clauses nf-bloquées, est un processus qui a été intégré au solveur LSAT (voir §2.2.1) afin de supprimer un maximum de clauses de la partie clausale restante après la détection des portes.

Pour être complet, sur les clauses nf-bloquées, il faut noter que les notions de clauses u-redondantes et nf-bloquées sont indépendantes :

- il existe des clauses nf-bloquées non u-redondantes (α_1 dans l'exemple précédent) ;
- il existe des clauses u-redondantes non nf-bloquées (toujours depuis l'exemple 9, si l'on ajoute les clauses $\alpha_6 = (a \vee \neg d)$ et $\alpha_7 = (\neg a \vee \neg d)$, α_1 n'est plus nf-bloquée mais devient u-redondante) ;
- il existe des clauses nf-bloquées et u-redondantes (cette fois, il faut ajouter $\alpha'_6 = (e \vee \neg d)$ et $\alpha'_7 = (\neg e \vee \neg d)$, pour obtenir α_1 nf-bloquée et u-redondante en même temps).

2.3.2 Production de sous-clauses à l'aide de la propagation unitaire

La propagation unitaire est un mécanisme fréquemment exploité dans l'algorithmique de SAT, comme heuristique [LA97], prétraitement [GMOS05], traitement local [DABC96], etc. Nous proposons de l'exploiter afin de produire des sous-clauses, c'est-à-dire des clauses subsumant les clauses de la formule. Pour ce faire, nous proposons une extension du graphe d'implications tel qu'il est utilisé dans les solveurs SAT modernes.

Le graphe d'implications associé à une CNF Σ et une interprétation partielle I, dépend à la fois de l'ordre d'affectation des littéraux de I mais également des choix réalisés lors des propagations unitaires. En effet, plusieurs clauses peuvent être à l'origine de l'implication par propagation unitaire d'un même littéral, comme l'illustre l'exemple suivant.

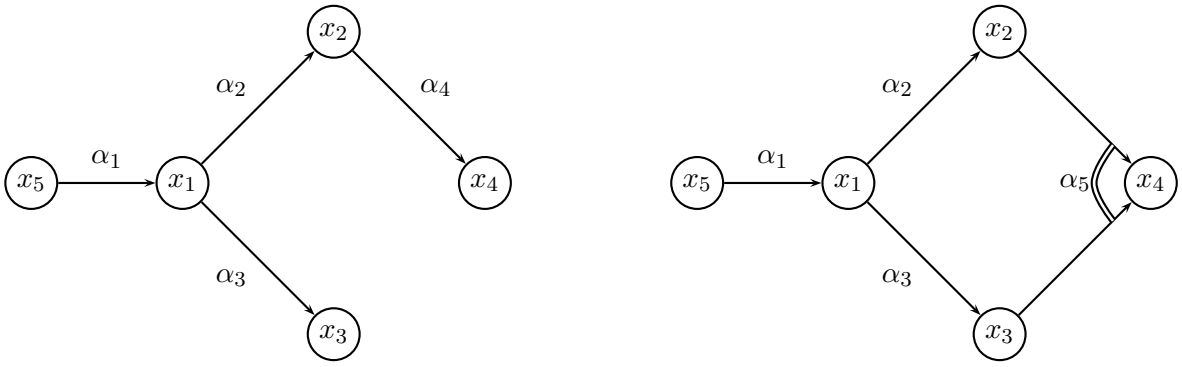
Exemple 10. Soit la formule $\Sigma = \{\alpha_1 = x_1 \vee \neg x_5, \alpha_2 = \neg x_1 \vee x_2, \alpha_3 = \neg x_1 \vee x_3, \alpha_4 = \neg x_2 \vee x_4, \alpha_5 = \neg x_2 \vee \neg x_3 \vee x_4\}$. Les figures 2.4a et 2.4b montrent deux graphes d'implications résultant de l'affectation de x_5 ; le littéral x_4 pouvant être produit de deux manières différentes.

Il en découle que les clauses produites par résolution et leur nombre dépendent du graphe d'implications considéré. Afin de s'affranchir de cet inconvénient, nous avons introduit la notion de *graphe d'implications complet* qui peut être vu comme l'union des graphes d'implications. Ce graphe permet de générer l'ensemble des clauses résolvantes. De cet ensemble, nous ne retenons que les clauses qui subsument au moins une clause de l'instance originale dans le but de la substituer.

Ce travail est le fruit d'une collaboration avec Sylvain DARRAS, Laure DEVENDEVILLE, Gilles DEQUEN, Richard OSTROWSKI et Lakhdar SAÏS. Dans **[DDD+05a, DDD+05b]**, nous envisageons une

[26] ℓ est un littéral pur dans ce cas.

FIGURE 2.4 : Dépendance des graphes d'implications à l'ordre de propagation



(a) x_4 impliqué par α_4

(b) x_4 impliqué par α_5

application dynamique, c'est-à-dire à chaque nœud de l'arbre de recherche, mais nous ne l'avons expérimenté qu'en prétraitement. Pour ce faire, chaque littéral de la formule est propagé successivement. Dans le cas où le processus de propagation génère la clause vide, on déduit le littéral opposé ; dans le cas contraire, le graphe d'implications est exploité pour réduire les clauses de Σ. Ce processus est répété jusqu'à ce qu'aucune nouvelle sous-clause ne puisse être produite. Les résultats expérimentaux montrent que non seulement cette technique permet de raccourcir un grand nombre de clauses mais également de fixer de nombreux littéraux. Cette simplification de la formule originale a un effet extrêmement bénéfique pour la résolution de l'instance.

Ce travail a été revisité et étendu dans le prétraitement appelé *vivification* proposé dans [PHL08].

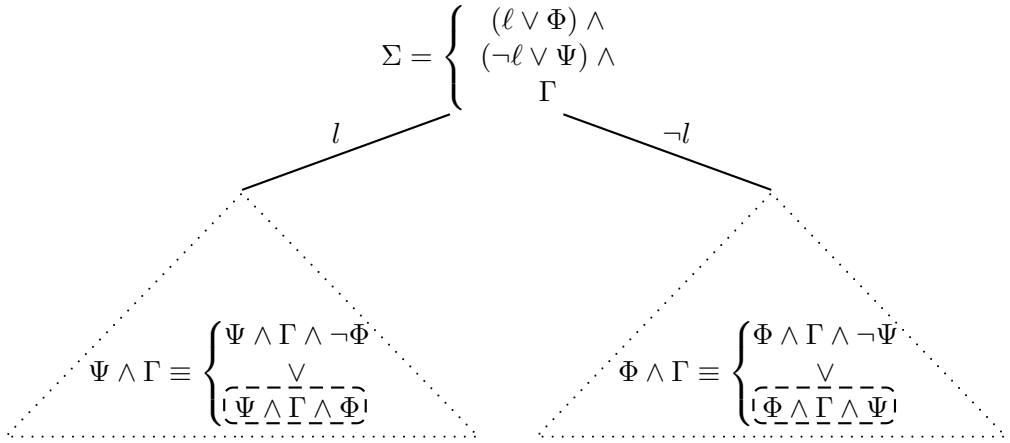
2.3.3 Recherche complémentaire

Les méthodes de simplification présentées cherchent à faciliter la résolution d'une CNF en lui apportant des modifications syntaxiques : retrait, ajout, substitution de clauses ou de littéraux. Nous nous intéressons dans cette section à une simplification de l'arbre de recherche sans manipulation syntaxique.

Dans **[OMSG03]**, le principe de recherche complémentaire initialement proposé par Paul PURDOM [Pur84] est revisité, analysé et appliqué de manière originale et productive. Le principe de la recherche complémentaire peut s'énoncer de la façon suivante : soient Σ une formule CNF, et $\ell \in \mathcal{V}_\Sigma$. Σ peut être réécrite de manière équivalente comme $(\ell \vee \Phi) \wedge (\neg\ell \vee \Psi) \wedge \Gamma$ où $\Phi = \{c \backslash \{\ell\} | \ell \in c \text{ et } c \in \Sigma\}$, $\Psi = \{c \backslash \{\neg\ell\} | \neg\ell \in c \text{ et } c \in \Sigma\}$ et $\Gamma = \{c | \{\ell, \neg\ell\} \cap c = \emptyset \text{ et } c \in \Sigma\}$. Sous cette écriture, Σ est satisfiable si et seulement si $\Sigma|_\ell$ est satisfiable ou $\Sigma|_{\neg\ell} \wedge \neg\Psi$ est satisfiable. En effet, comme l'illustre la figure 2.5, s'il est prouvé que $\Sigma|_l$ ne peut pas admettre de modèle alors, la formule $\Psi \wedge \Phi \wedge \Gamma$ ne peut pas être satisfiable et il est inutile d'explorer de nouveau cette formule dans l'autre partie de l'arbre de recherche.

Comme le fait remarquer Paul PURDOM, la recherche complémentaire joue un rôle important d'un point de vue théorique sauf que pour être exploitée il faut ajouter explicitement des contraintes obtenues à partir de la négation d'une sous-formule CNF dont la transformation en formule CNF est coûteuse. Cette contrainte de mise en œuvre pratique a été citée par Giorgio GALLO et Giampaolo URBANI [GU89] : « *Purdom's branching criterion succeeds in reducing the size of the search tree but a price must be paid. In fact, the formula must be transformed into the standard form of set of clauses, which might be quite costly.* ». Répondant à cette limitation pratique, nous proposons une mise en œuvre efficace du principe de recherche complémentaire dans les algorithmes de type DPLL. En outre, cette mise en œuvre introduit une nouvelle manière de dériver des littéraux unitaires.

FIGURE 2.5 : Principe de la recherche complémentaire



Partant simplement du fait que $\neg\Psi$ est vrai si et seulement si Ψ ne l'est pas, il est possible d'éviter aisément l'ajout de $\neg\Psi$. En effet, lorsque l'ensemble Ψ est satisfait (c'est-à-dire que toutes les clauses contenant $\neg\ell$ sont *sursatisfaites*[27]) la formule $\neg\Psi$ est insatisfiable. Par conséquent, il suffit simplement d'empiler des pointeurs sur les clauses où $\neg l$ apparaît et de vérifier au cours de la recherche dans le sous-arbre $\Sigma|_{\neg\ell}$ si ces clauses sont sursatisfaites. Ce test est réalisé en maintenant pour chaque clause le nombre de littéraux qui la satisfont. Ainsi la formule $\neg\Psi$ est considérée sans être ajoutée explicitement à $\Sigma|_{\neg\ell}$.

Un autre aspect important est la possibilité de déduire des littéraux unitaires. Lorsque toutes les clauses de Ψ sont satisfaites exceptée une, il est alors possible d'impliquer la négation des littéraux non affectés de cette clause. Dans **[OMSG03]**, nous montrons les gains obtenus quant à la résolution de nombreuses instances par un algorithme DPLL, lorsque le principe de recherche complémentaire et la nouvelle règle de production de littéraux unitaires sont appliqués.

[27] Une clause α est dite *sursatisfaite* pour une interprétation I si et seulement si $|I \cap \alpha| > 1$.

Sommaire

Chapitre

3

Au-delà de SAT

De nombreux travaux sur SAT reposent sur la localisation de contraintes difficiles à satisfaire ou souvent falsifiées qui sont heuristiquement jugées comme responsables de la non-satisfiabilité de l'instance. C'est donc naturellement que nous avons essayé d'appliquer la même stratégie pour la résolution de problèmes autour de SAT.

Les premières contributions discutées dans ce chapitre concernent une application directe de cette stratégie à la recherche de noyaux incohérents, encore appelés MUS, au sein d'une instance de SAT. Puis ce problème est transposé au niveau des CSP où des MUC et des MUST sont recherchés. La stratégie décrite ci-dessus et l'algorithmique s'y rapportant sont ensuite appliqués à la résolution de problèmes exprimés à l'aide de logiques non monotones ou du premier ordre. Enfin nous concluons ce chapitre par la présentation de notre contribution aux problèmes de la compilation de bases de connaissances et à la résolution d'instances QBF.

3.1 Calcul de noyaux : cadre propositionnel

Lorsque le résultat du test de satisfiabilité d'une formule propositionnelle s'avère positif, la plupart des solveurs fournissent un modèle de la formule. En revanche, lorsqu'une méthode complète établit qu'une formule n'admet pas de modèle, elle ne délivre généralement aucune explication quant aux raisons de ce résultat. Or, il est possible que seules quelques clauses soient réellement contradictoires, rendant l'ensemble de la formule incohérente. De ce point de vue, la localisation au sein de la formule

ALGORITHME 3.1 : AOMUS

input : Σ une CNF
output : l'approximation d'un MUS de Σ

```
1 begin
2     stack ⟵ ∅ ;
3     while (LocalSearchWithScoredCriticalClauses(Σ) ne trouve pas de modèle)
      do
4         push(stack, Σ) ;
5         Σ ⟵ Σ \ {α | score(α) < (min(score) + #flips/|Σ|)} ;
6     end
7     repeat Ω ⟵ pop(stack) until CDCL(Ω) ;
8     return Ω ;
9 end
```

des raisons de sa non-satisfiabilité peut se révéler utile dans nombre de domaines. Une telle information, de nature diagnostique, peut être fournie par la localisation de sous-formules minimallement non satisfiables (ou MUS pour « *Minimally Unsatisfiable Subformulae* ») d'une CNF.

3.1.1 Extraction d'un MUS

De nombreux algorithmes ont été proposés ces dernières années pour extraire une sous-formule non satisfiable d'une CNF. La plupart de ces algorithmes n'assurent pas la minimalité. Parmi ces approches, citons la méthode *adaptative* [Bru03], *amuse* [OMA+04], *zcore* [ZM03], etc. Concernant la minimisation de la formule, des techniques dédiées ont été développées comme *zminimal* [ZM03], *mup* [Hua05] ou *miniunsat* [vMW08]. Cependant l'efficacité de ces algorithmes de minimisation dépend fortement de l'approximation fournie en entrée. Nous nous sommes intéressés dans un premier temps aux algorithmes d'approximation et avons proposé une utilisation originale et efficace de la recherche locale pour localiser les sources d'incohérence.

Dans **[MSG98]** nous utilisons la recherche locale à chaque décision de DPLL pour déterminer la variable à affecter. La variable retenue est celle qui est apparue le plus souvent dans les clauses les plus falsifiées durant la recherche locale (voir §2.1.3). L'hypothèse qui a conduit à cette heuristique est : « *les clauses les plus souvent falsifiées appartiennent aux MUS de la CNF* ». L'un des objets de la thèse de David ANSART [Ans05] que j'ai co-encadrée, portait sur le raffinement de cette hypothèse. Cette étude a permis d'introduire la notion de *clauses critiques*. Cette notion a très largement été enrichie et exploitée dans la thèse de Cédric PIETTE [Pie07] que j'ai également co-encadrée.

Une clause α d'une CNF Σ est dite *critique* pour une interprétation I si et seulement si $\alpha \cap I = \emptyset$ et $\forall \ell \in \alpha, \exists \alpha' \in \Sigma$ telle que $\alpha' \cap I = \{\overline{\ell}\}$. Autrement dit, α est falsifiée par I et il existe, pour chaque littéral ℓ de α, une clause unisatisfaite par $\overline{\ell}$. Ces clauses unisatisfaites sont dites *liées* à α.

Exemple 11. Soient la formule $\Sigma = \{(a \vee \neg b),\ (\neg b \vee c),\ (\neg a \vee \neg b \vee \neg c),\ (\neg a \vee b \vee \neg c),\ (a \vee c,\ a \vee b),\ (\neg a \vee b \vee c)\}$ et l'interprétation $I = \{a,\ \neg b,\ \neg c\}$. Sous cette interprétation, la clause falsifiée $(\neg a \vee b \vee c)$ est critique. Ses clauses liées sont $(a \vee b)$ et $(a \vee c)$ pour $\neg a$, $\{\neg b \vee c\}$ pour b et $(\neg a \vee b \vee \neg c)$ pour c.

Les clauses critiques présentent des propriétés intéressantes quant à l'extraction de MUS. Dans **[GMP06a]**, en collaboration avec Éric GRÉGOIRE et Cédric PIETTE, nous montrons que pour chaque clause d'un MUS, il existe une interprétation qui la rend critique. Par ailleurs, lors de l'exécution d'un

ALGORITHME 3.2 : OMUS

input : Σ une CNF
output : un MUS de Σ

```
1 begin
2 |  Ω ⟵ AOMUS(Σ) ;
3 |  foreach α ∈ Ω do
4 |  |  if (CDCL(Ω \ {α})) then Ω ⟵ Ω \ {α} ;
5 |  end
6 |  return Ω ;
7 end
```

algorithme de type recherche locale, lorsqu'un minimum local est atteint, nous sommes assurés que toutes les clauses falsifiées sont critiques et donc qu'au moins une clause par MUS est critique. De ces propriétés, nous proposons un algorithme nommé AOMUS (« *Approximate One MUS* ») et basé sur la recherche locale pour approximer un MUS. Le principe est d'attribuer un score à chaque clause ; ce score est augmenté dès lors qu'une clause est critique ou liée à une clause critique pour une interprétation parcourue par la méthode de recherche locale. À chaque itération, les clauses ayant obtenu les plus faibles scores sont supprimées jusqu'à ce que la recherche locale délivre un modèle. Les clauses précédemment supprimées sont de nouveau ajoutées jusqu'à ce que l'instance n'admette plus de modèle. AOMUS est détaillé dans l'algorithme 3.1.

L'approximation fournie par AOMUS peut ensuite être minimisée avec une des méthodes citées précédemment. Nous utilisons un méthode dite *destructrice* qui consiste à tester chaque clause et à la supprimer si la CNF privée de cette clause demeure non satisfiable. Dans **[GMP08a]**, nous discutons des différentes méthodes de minimisation. L'algorithme OMUS (« *One MUS* »), présenté dans l'algorithme 3.2 calcule un MUS, d'une CNF fournie en entrée de l'algorithme.

Les expérimentations conduites sur AOMUS montrent que l'approche est extrêmement compétitive fournissant, dans la très grande majorité des cas, une meilleure approximation (en terme de taille) comparativement aux autres méthodes de la littérature et ceci plus rapidement.

3.1.2 Extraction d'un MUS particulier

Les algorithmes permettant l'extraction d'un MUS d'une CNF retournent n'importe quel MUS de l'instance. Or dans certaines applications, il est nécessaire d'identifier un MUS particulier, comme le plus petit MUS en terme de taille. La recherche du plus petit MUS a fait l'objet de plusieurs études notamment dans [LMS04, MLA+05, ZLS06]. Dans **[GMP09a]** nous nous sommes intéressés au calcul d'un MUS contenant au moins une clause d'un sous-ensemble de clauses donné.

Nous proposons dans **[GMP09a]** une approche originale pour répondre à ce problème, sans calculer *tous* les MUS de Σ. Pour contourner la très haute complexité dans le pire cas de cette requête, le problème est exprimé dans un cadre à gros grains où des *clusters* de clauses de la CNF sont formés, puis considérés comme les éléments de base de la formule et examinés selon leur niveau de conflits mutuels. Le cadre est ensuite progressivement raffiné en divisant les clusters les plus prometteurs et en élaguant ceux qui ont été prouvés inutiles jusqu'à ce que la quantité de ressources allouées soit épuisée ou jusqu'à ce qu'une solution soit retournée. Le niveau de conflits mutuels entre clusters est mesuré grâce à une forme de valeur de SHAPLEY [Sha53], célèbre dans la communauté de la théorie des jeux et appliquée à la mesure de l'incohérence dans [HK08].

3.1.3 Calcul de tous les MUS

Si la méthode proposée précédemment ne nécessitait pas le calcul de tous les MUS, il est parfois nécessaire de calculer cet ensemble dont la taille peut être exponentielle en fonction du nombre de clauses de la CNF. Peu de méthodes et surtout peu de méthodes (relativement) efficaces ont été proposées, citons tout de même [dlBSW03, BS05]. La méthode *camus* [LS05, LS08], proposée récemment par Mark LIFFITON et Karem SAKALLAH, donne les résultats les plus probants. Elle repose (comme d'autres méthodes) sur l'exploitation de la dualité entre les MUS et les MSS. Cette méthode se décompose en deux étapes distinctes :

- le calcul de l'ensemble des MSS. Pour cela, la méthode utilise un solveur SAT moderne et recherche successivement tous les MSS de tailles $|\Sigma| - 1$, $|\Sigma| - 2$, etc.
- à partir des MSS calculés et plus précisément des CoMSS, l'ensemble des MUS est produit en cherchant tous les ensembles intersectants minimaux. Ce problème est équivalent au calcul des transversaux minimaux d'un hypergraphe pour lequel de nombreux algorithmes ont été proposés.

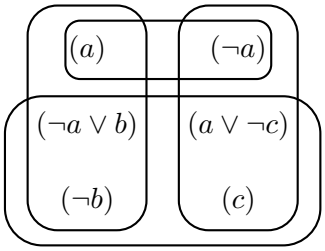
Dans **[GMP07a, GMP09b]** nous nous sommes intéressés à améliorer la première phase de l'algorithme. Nous avons proposé une amélioration qui repose sur une hybridation de la méthode calculant les MSS avec une méthode de recherche locale. La méthode de recherche locale fournit à la méthode complète des CoMSS candidats trouvés au cours de l'exploration stochastique de l'espace de recherche. Afin de sélectionner des « bons » candidats, le concept de clauses critiques est une nouvelle fois utilisé. En effet, pour une interprétation, si toutes les clauses falsifiées ne sont pas critiques, alors clairement ces clauses ne peuvent pas constituer un CoMSS. D'autre part, il est inutile de conserver un candidat qui serait un sur-ensemble d'un candidat nouvellement calculé. Pour ce faire, nous avons adapté la technique proposée dans [Zha05], pour le calcul des ensembles de clauses subsumées, afin de n'enregistrer que les CoMSS candidats minimaux. Une fois l'ensemble de candidats calculés, ceux-ci sont directement fournis à *camus* qui peut ainsi éviter le calcul de certains de ces CoMSS. En effet, lorsque l'algorithme calcule les MSS de taille $|\Sigma| - i$, il peut directement ajouter tous les CoMSS candidats qui ne sont pas des sur-ensembles des CoMSS déjà calculés (c'est-à-dire ceux de taille inférieure à i). Cela permet d'éviter des tests « NP-complets » et « CoNP-complets » et offre donc un gain substantiel confirmé par les expérimentations réalisées.

3.1.4 Approximation de l'ensemble des MUS

Lorsque le calcul de l'ensemble exact des MUS s'avère hors de portée des algorithmes, par exemple parce que l'ensemble des CoMSS est trop grand, il peut être intéressant de disposer d'un algorithme approximant cet ensemble.

Dans **[GMP07b]**, nous proposons un algorithme ASMUS (« *Approximate Set of MUS* »), basé sur la procédure OMUS pour fournir une approximation de l'ensemble des MUS d'une CNF. Partant du constat qu'il suffit de supprimer une clause à un MUS pour qu'il devienne satisfiable, une approche naïve consiste à supprimer une clause de la CNF appartenant au MUS délivré par OMUS. Un nouveau MUS peut alors être recherché ; ce processus est itéré jusqu'à ce que la CNF réduite devienne satisfiable. Une telle approche délivre l'ensemble exact de MUS si et seulement si chaque paire de MUS possède une intersection vide. Cependant, les MUS ont souvent une intersection non vide et supprimer une clause de cette intersection supprime les deux MUS. Nous proposons une heuristique qui consiste à supprimer les clauses pour lesquelles le nombre minimal de clauses fausses lorsqu'elles étaient falsifiées durant la recherche locale, est maximal. Cette heuristique repose sur le fait que lorsque l'on se trouve dans un optimum global, c'est-à-dire une solution pour le problème MAXSAT, les clauses fausses appartiennent aux intersections des MUS. Choisissant les clauses suivant le critère inverse, nous espérons supprimer un minimum de MUS lors du retrait de la clause. ASMUS est détaillé dans l'algorithme 3.3. Il est clair

FIGURE 3.1 : Ensemble de MUS impossible à calculer via ASMUS



ALGORITHME 3.3 : ASMUS

input : Σ une CNF
output : une approximation de l'ensemble des MUS de Σ

begin
  $S_{MUS} \longleftarrow \emptyset$;
  while (CDCL(Σ) = **false**) **do**
    $\Omega \longleftarrow$ OMUS(Σ) ;
    $S_{MUS} \longleftarrow S_{MUS} \cup \{\Omega\}$;
    $\alpha \longleftarrow$ la clause de Ω dont le nombre minimum de clauses falsifiées en même temps qu'elle durant la recherche est maximal ;
    $\Sigma \longleftarrow \Sigma \setminus \{\alpha\}$;
  end
  return S_{MUS} ;
end

que cet algorithme fournit une approximation et que dans certaines configurations, il lui est impossible de calculer tous les MUS. Par exemple, sur la formule $\Sigma = \{(a), (\neg a), (\neg a \vee b), (\neg b), (a \vee \neg c), (c)\}$, il ne pourra délivrer qu'au maximum 3 des 4 MUS de la formule, la première suppression de clause supprimant au minimum 2 MUS de la formule, comme le montre la figure 3.1 où les MUS sont regroupés.

Dans **[GMP06b]**, l'ensemble calculé par ASMUS est appelé *couverture incohérente* (« *inconsistant cover* »). Formellement, elle se définit comme un ensemble de MUS tel que l'instance privée de l'union des MUS de la couverture est satisfiable. La couverture est dite *stricte* si et seulement si l'intersection deux à deux de chacun des MUS est vide. Une couverture stricte permet de calculer les causes indépendantes, c'est-à-dire non corrélées, d'incohérence au sein d'une même formule. Ceci peut avoir du sens dans certaines applications comme par exemple, la détection de pannes simultanées d'un système complexe. L'algorithme ICMUS (voir algorithme 3.4) calcule une telle couverture.

3.2 Calcul de noyaux : cadre CSP

Tout comme dans le cadre propositionnel, lorsqu'un solveur CSP prouve qu'une instance n'admet pas de solution, aucune information n'est délivrée quant à la raison de l'incohérence. Or celle-ci peut-être due à un sous-ensemble de contraintes qui rend l'ensemble du réseau de contraintes en totalité incohérent. Le calcul des sous-ensembles de contraintes minimalement incohérents (MUC) permet de localiser les éléments en conflit afin, par exemple, de diagnostiquer la cause de l'incohérence et éventuellement la réparer.

ALGORITHME 3.4 : ICMUS

input : Σ une CNF
output : une couverture incohérente stricte de Σ

```
1 begin
2     IC_MUS ⟵ ∅ ;
3     while (CDCL(Σ) = false ) do
4         Ω ⟵ OMUS(Σ) ;
5         IC_MUS ⟵ IC_MUS ∪ {Ω} ;
6         Σ ⟵ Σ \ Ω ;
7     end
8     return IC_MUS ;
9 end
```

Plusieurs approches ont par le passé été proposées dans le but de détecter des MUC. Tout d'abord, les approches générales de [HL99] et [dlBSW03], proposées dans d'autres contextes, permettent l'obtention de tous les MUC par l'exploration d'un arbre « *CS* ». Celui-ci consiste à énumérer et tester la cohérence des sous-problèmes du CSP traité pour en extraire l'ensemble de ses MUC. Malheureusement, l'explosion combinatoire du nombre de sous-problèmes d'un CSP ne permet à cette méthode que de pouvoir traiter de très petits problèmes en pratique.

Plus spécifiquement au cadre CSP, plusieurs travaux visent à identifier des ensembles (minimaux) conflictuels de contraintes afin d'effectuer des retours en arrière intelligents, tels que le « *dynamic backtracking* » [Gin93, JDB00] ou le « *conflict-based backjumping* » [Pro93]. Toutefois, seules quelques méthodes spécifiques à l'extraction d'un MUC ont été proposées. Notons par exemple les approches de [MT02] et *quickexplain* [Jun01, Jun04] qui permettent d'obtenir une explication de l'incohérence basée sur les préférences de l'utilisateur. Mentionnons également la bibliothèque *palm* [JB00] de la plate-forme de programmation par contraintes *choco* [Lab00], qui est un outil permettant par exemple d'expliquer pourquoi il n'existe aucune solution contenant une valeur particulière à un CSP donné. De plus, en cas d'inconsistance du problème, *palm* est capable d'en extraire un sous-problème non satisfiable, mais la minimalité de celui-ci n'est pas garantie. L'approche baptisée *dc(wcore)*, introduite dans [HLSB06] améliore une précédente méthode [BDTW93] introduite dans le contexte du diagnostic de pannes. Elle a été prouvée plus efficace que le système *quickxplain*, et semble être une des méthodes les plus performantes en pratique pour extraire un MUC. Notre contribution à l'algorithmique d'extraction de MUC porte sur l'amélioration de cette technique.

3.2.1 Extraction d'un MUC

La méthode *dc(wcore)* se décompose en deux étapes, nommées *wcore* et *dc*. La première extrait un ensemble conflictuel de contraintes sans que la minimalité ne soit garantie, alors que la seconde délivre un MUC à partir de cet ensemble. *wcore* utilise l'exploration faite par un algorithme classique de satisfaction de contraintes de type « *branch and bound* » et marque toutes les contraintes qui ont été utiles pour réduire la taille d'un domaine d'une variable lors du maintien de l'arc-consistance. Ces contraintes sont marquées comme *actives*. À la terminaison de l'algorithme, les contraintes non marquées peuvent être supprimées et l'algorithme fournit un sous-ensemble de contraintes en conflit. Il est évident que le sous-problème extrait par cette approche peut différer en fonction des affectations partielles traversées, qui sont dirigées par l'heuristique de choix de variables. En pratique, *wcore* jouit de l'efficacité de l'heuristique dynamique $dom/wdeg$ [BHLS04] qui permet de concentrer la recherche sur des zones surcontraintes. Cette opération peut être répétée sur le sous-problème obtenu afin d'en extraire un éventuel

nouveau sous-problème. De cette façon, *wcore* consiste en une boucle où sont itérés des appels à un algorithme complet de recherche tant que le nombre de contraintes actives décroît. De plus, les compteurs de l'heuristique $dom/wdeg$ sont conservés d'un appel à l'autre, de manière à se concentrer progressivement sur une petite zone de l'espace de recherche, et ainsi réduire le nombre de contraintes actives. En pratique, la conservation de ces compteurs se montre extrêmement efficace pour la découverte de petits ensembles conflictuels de contraintes.

Notre première contribution à la recherche d'un MUC dans le cadre CSP vise à améliorer les performances de *wcore*. Pour ce faire, nous retardons le mécanisme de retour-arrière, dans un double objectif : identifier plus de contraintes actives et affiner les poids utilisés dans l'heuristique $dom/wdeg$. Alors que l'algorithme utilisé effectue un retour-arrière dès qu'une contrainte est violée, nous préconisons de poursuivre le filtrage en cours. Ceci permet de ne pas choisir arbitrairement le premier échec rencontré alors que des échecs situés au même niveau de propagation peuvent être obtenus. Cette stratégie a déjà été appliquée à d'autres contextes que la recherche d'un MUC, comme par exemple dans [SV94]. Les résultats expérimentaux montrent que le surcoût de calculs engendré par cette stratégie est compensé par la qualité du sous-ensemble de contraintes obtenu qui devient plus simple à minimiser.

Notre seconde contribution porte sur l'algorithme de minimisation. Dans [HLSB06] les auteurs préconisent une stratégie totalement dichotomique pour réduire le sous-ensemble de contraintes obtenu par *wcore* en un MUC. Nous pensons que l'hypothèse, déjà vérifiée dans le cadre propositionnel, consistant à dire que les contraintes les plus souvent violées ont plus de chances d'appartenir à un MUC, permet à une approche de type « destructrice »[28] de délivrer un MUC rapidement. Même si cette approche dans le pire des cas possède une complexité algorithmique plus importante que la stratégie dichotomique, elle devrait fournir en pratique de bons résultats. En effet, si les contraintes sont testées dans l'ordre décroissant du nombre de fois où elles ont été violées, nous pouvons espérer, selon l'hypothèse faite, qu'un maximum des premiers tests soient des tests prouvant la satisfiabilité de l'ensemble, tests souvent moins coûteux que des tests prouvant l'absence de solution. Ainsi nous préconisons une stratégie qui combine l'approche dichotomique et l'approche destructive où la dichotomie est utilisée uniquement pour délivrer le premier sous-ensemble de contraintes. Les expérimentations conduites ont confirmé que cet algorithme de minimisation se montre plus efficace que la stratégie dichotomique. Par ailleurs, cette stratégie combinée à l'approche jusqu'au-boutiste de *wcore* offre de bien meilleures performances que *DC(wcore)*.

Ces travaux réalisés conjointement avec Éric GRÉGOIRE, Cédric PIETTE et Lakhdar SAÏS ont été publiés dans **[GMPS06, GMP08b]**.

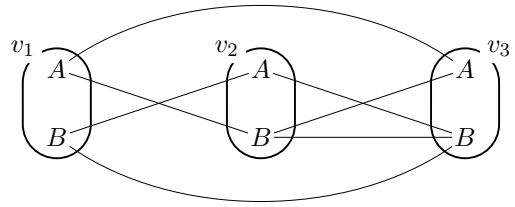
3.2.2 Extraction d'un MUST

Restaurer la satisfiabilité d'un CSP peut être effectué à travers la réparation de chacun de ses MUC. Une façon naturelle de « casser » l'incohérence due à un MUC passe par la suppression de l'une de ses contraintes. Cependant, une telle suppression peut apparaître comme un acte destructeur. Alternativement, il peut être préférable d'*affaiblir* une ou plusieurs contraintes, plutôt que de les supprimer. Une façon d'effectuer une telle opération est de fournir à l'utilisateur certains tuples interdits tels que leur autorisation suffit à restaurer la satisfiabilité.

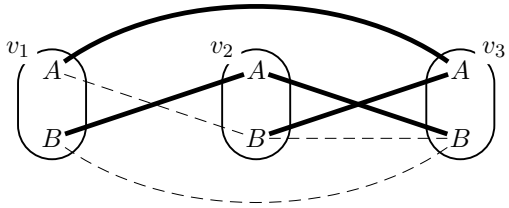
Dans cet objectif, dans **[GMP07c]**, en collaboration avec Éric GRÉGOIRE et Cédric PIETTE, nous introduisons le concept de MUST. Un MUST (« *Minimally Unsatisfiable Set of Tuples* »), d'un CSP non satisfiable, est un ensemble $\mathcal{T}$ non satisfiable de tuples interdits tel que tout sous-ensemble de $\mathcal{T}$ est satisfiable. On retrouve avec les MUST, les problèmes inhérents aux ensembles minimalement incohérents.

[28] Cette approche consiste à tester si pour chaque contrainte, le CSP est toujours non satisfiable une fois cette contrainte retirée. Dans l'affirmative, la contrainte est supprimée du CSP et les tests se poursuivent sur ce nouveau CSP ; dans la négative, la contrainte est conservée.

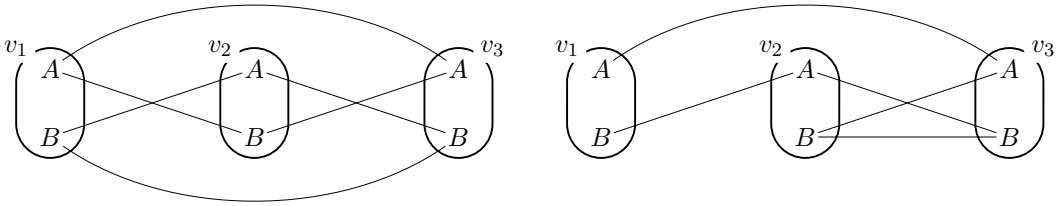
FIGURE 3.2 : MUST et tuples partagés



(a) Graphe de micro-structure



(b) Tuples partagés



(c) Graphes de micro-structure des deux MUST

Il ne suffit donc pas de supprimer un tuple d'un MUST d'un problème pour restaurer la cohérence d'un CSP. En effet, on peut prouver facilement qu'il existe dans le pire cas un nombre exponentiel de MUST au sein d'un CSP n'admettant pas de solution, et que ces MUST peuvent avoir une intersection vide. Par ailleurs, tout MUC possède au moins un MUST, mais il peut exister des MUST faisant intervenir des contraintes n'appartenant à aucun MUC !

Malgré ces résultats négatifs, les MUST forment bien un concept viable pour exprimer les causes de l'incohérence d'un CSP. En effet, nous avons montré que si P est un MUC de m contraintes, alors il existe au moins m tuples tels qu'autoriser l'un d'entre eux restaure la satisfiabilité de P. Ces tuples sont appelés les *tuples partagés*. La figure 3.2 représente le graphe de micro-structure du MUC $\langle\{v_1, v_2, v_3\}, \{(v_1 \neq v_2), (v_2 + v_3 = 2A), (v_1 = v_3)\}\rangle$ avec $dom(v_i) = \{A, B\}$ les 2 MUST qu'il contient et les tuples partagés. Afin de restaurer la cohérence d'un MUC en relâchant des contraintes, il suffit donc d'identifier l'ensemble de ces tuples partagés. À partir d'un CSP, l'algorithme procède donc en deux étapes : l'identification d'un MUC à l'aide de la technique présentée dans la section précédente et le calcul d'un MUST (et de tuples partagés) au travers l'algorithme OMUS (voir 3.1.1) développé dans le cadre propositionnel. En effet, nous avons établi que les concepts de MUST et MUS correspondent dès lors que l'on utilise un codage de type « *direct encoding* » [dK89] sans les clauses « *at most one* ». Par ailleurs, durant la recherche d'un MUS dès lors qu'une clause est la seule à être falsifiée pour une interprétation donnée, cette clause est marquée car elle correspond forcément à un des tuples partagés du MUC. Les expérimentations conduites ont montré qu'il était possible d'extraire des MUST, souvent très petits, comparativement à l'ensemble initial de tuples du CSP et que l'approche était capable en pratique de fournir les tuples à relâcher (les tuples partagés) afin de restaurer la cohérence sans supprimer de contraintes.

3.3 Algorithmique de SAT appliqué aux logiques non monotones et du premier ordre

3.3.1 Méthode complète pour des bases de connaissances propositionnelles non monotones

Nous avons proposé une nouvelle approche pour calculer de façon effective des inférences au sein d'une logique propositionnelle non monotone simple, fondée sur un concept de modèles préférés. L'approche que nous proposons est originale à deux égards au moins. Premièrement, elle se base sur des techniques de recherche locale tout en préservant la complétude logique. En second lieu, elle est efficace d'un point de vue expérimental pour une classe importante de grandes bases de connaissances non monotones. Plus précisément, nous étendons à un cadre non monotone des résultats heuristiques liés à la résolution pratique du problème de satisfiabilité d'un ensemble de clauses (SAT). De fait, la procédure de preuve employée se base sur l'utilisation d'une méthode de recherche locale pour SAT et sur une heuristique efficace lorsque cette recherche locale n'a pu exhiber un modèle. Nous l'employons au sein d'un formalisme de représentation permettant des règles de raisonnement par défaut avec priorités exprimées à l'aide de propositions d'anormalités à la McCarthy. Un domaine d'applications typique concerne les formes de raisonnement révisable qui peuvent être tenues au sujet d'un modèle d'un système ou d'un appareillage complexe, où les propositions d'anormalité sont utilisées pour représenter les défaillances possibles de composants et où un nombre limité de pannes peuvent survenir simultanément **[MSG97a, GMS98, GMS02]**.

3.3.2 Coopération consistante et non-monotonie

Notre expérience dans le cadre de la détection et de la localisation des inconsistances nous a conduit à envisager l'extension et l'application de ces techniques pour résoudre les problèmes liés à la fusion et à l'interaction des bases de connaissances dans un cadre de travail coopératif. En effet, le problème de maintien de la consistance lorsque l'on fait interagir différentes sources d'informations est fondamental dans de nombreuses applications : bases de données distribuées, systèmes multi-agents, travail coopératif, etc.

Dans ce cadre, nous avons montré que les techniques que nous avons développées dans le cadre de SAT peuvent être utilisées pour localiser les inconsistances qui peuvent apparaître lors de la fusion de bases de connaissance. Une partie de cette étude est publiée dans le journal « *International Journal of Cooperative Information Systems* » **[MSG97b]**.

3.3.3 Application au premier ordre fini

Les progrès obtenus dans la résolution de SAT conduisent naturellement à l'extension et l'application de ces techniques dans le cadre de la logique du premier ordre fini. Une approche naïve consiste à appliquer les algorithmes de SAT sur une instanciation complète de la formule du premier ordre. Cette méthode peut provoquer une explosion combinatoire. Afin de l'éviter, il est possible d'exploiter le schéma d'instanciation proposé dans [BSG01] qui s'est avéré viable pour le test de cohérence d'une base de connaissances du premier ordre fini (restreint aux interprétations de Herbrand) construite incrémentalement.

Dans **[GM02]**, en collaboration avec Éric Grégoire, nous avons revisité cette technique pour des bases stratifiées suivant un pré-ordre indiquant une préférence sur les connaissances. Nous avons notamment montré qu'il était possible de calculer une sous-base maximalement cohérente préférée, comme définie dans [BCD+93].

3.4 Compilation des bases de connaissances

Afin de pallier la complexité théorique de la déduction logique en calcul propositionnel, plusieurs approches ont été proposées. Elles s'appuient sur différents principes, en particulier :

- restriction du langage utilisé, les clauses de HORN, par exemple, ou de la relation de déduction employée comme par exemple la résolution bornée ;
- approximation de la base de connaissances ou de la relation de déduction ;
- compilation. L'idée est de construire une structure de données $\widehat{\Sigma}$ (une compilation de la base initiale Σ) qui se comporte comme Σ pour l'interrogation mais nécessite seulement un temps polynomial (en fonction de $|\widehat{\Sigma}|$).

En collaboration avec Yacine BOUFKHAD, Éric GRÉGOIRE, Pierre MARQUIS et Lakhdar SAÏS, nous nous sommes particulièrement intéressés à ce dernier principe. Nous avons proposé une approche originale de compilation préservant l'équivalence logique avec la base de connaissances initiale. Cette approche est appelée *couverture traitable*[29]. Elle a été formalisée dans un cadre général. Deux cas spécifiques ont été considérés :

1. les interprétations partielles sont utilisées pour « raboter » ou simplifier la base de connaissances en un ensemble de formules traitables (simplifications traitables) **[MM96]** ;
2. les interprétations partielles sont utilisées de manière à dériver un ensemble de formules HORN-renommables (hyper-impliquants).

Dans le second cas, nous avons prouvé que la taille de la couverture est strictement inférieure à celle d'une couverture par les impliquants premiers. Il faut noter qu'en pratique les formules traitables ne sont pas représentées, seules les interprétations partielles (obtenues par la procédure de DPLL) nous permettant de les obtenir sont réellement sauvegardées. Des résultats expérimentaux ont montré l'efficacité d'une telle approche par rapport à l'interrogation directe de la base de connaissances. Un gain exponentiel en temps et en espace est obtenu par rapport à l'approche par impliquants premiers [CC96, Sch96]. Ce travail a été publié dans **[BGM+97]**.

3.5 QBF et symétries

Depuis quelques années, la résolution de formules booléennes quantifiées (QBF) est un domaine de recherche en plein essor. Plusieurs facteurs contribuent à cet intérêt. D'une part, de nombreux problèmes en intelligence artificielle (planification, raisonnement non monotone, vérification formelle, etc.) peuvent se réduire à des formules booléennes quantifiées. D'autre part, les progrès réalisés sur la résolution pratique de SAT permettent de résoudre des instances de plus en plus grandes et difficiles. C'est pourquoi de nombreux solveurs QBF ont vu le jour ces dernières années, citons par exemple *qube* [GNT01], *quantor* [Bie05], *skizzo* [Ben05], etc. La plupart étendent des résultats obtenus sur la résolution d'instances de SAT. Notre contribution à la résolution de formules booléennes quantifiées porte sur la définition, la recherche et l'exploitation des symétries pour ces formules.

De nombreuses tâches et problèmes combinatoires contiennent des symétries. La résolution de tels problèmes conduit à répéter, parfois de nombreuses fois, l'étude de situations ou de sous-problèmes équivalents. L'exploitation des symétries permet une réduction significative de l'espace de recherche. Ce paradigme a fait l'objet de nombreuses études, notamment sur SAT et CSP. Nos travaux portent sur l'extension de ce paradigme aux QBF.

Formellement, une symétrie σ pour une QBF $\mathcal{Q}_1 X_1, \mathcal{Q}_2 X_2, ..., \mathcal{Q}_n X_n \Psi$ est une permutation sur $\mathcal{L}_\Psi$ telle que $\forall \ell \in \mathcal{L}_\Psi, \sigma(\neg\ell) = \neg\sigma(\ell)$, $\sigma(\Psi) = \Psi$ et $\forall i \in [1..n]\sigma(X_i) = X_i$. Notons que chaque symétrie

[29] Une couverture traitable est une disjonction de formules pour lesquelles la déduction logique peut être réalisée en temps polynomial

sur la QBF est également une symétrie sur la CNF Ψ. Par contre, la réciproque est fausse. La détection des symétries peut donc s'opérer comme dans le cadre SAT sauf qu'il faut s'assurer que les permutations respectent les ensembles préfixes. Pour ce faire, nous étendons le concept de graphe coloré proposé dans [ARMS02] pour la détection de symétries au travers de la recherche d'automorphismes de graphes. Cette extension consiste à colorier les sommets littéraux appartenant au même ensemble préfixe par la même couleur, afin de n'autoriser les permutations qu'à l'intérieur de ces ensembles. Dans le cadre de SAT, l'exploitation des symétries se fait le plus souvent par l'ajout de prédicats cassant les symétries, appelés les « *Symmetry Breaking Predicates (SBP)* ». Dans le contexte des formules booléennes quantifiées, il est impossible d'ajouter ces prédicats à la matrice. En effet, ces prédicats peuvent se présenter sous la forme d'une clause où tous les littéraux sont universellement quantifiés, ce qui aurait pour effet de rendre la formule non valide. Ces prédicats ne sont donc pas ajoutés à la matrice mais stockés dans une CNF annexe sur laquelle les propagations unitaires sont maintenues. Lors de la résolution d'une QBF, chaque affectation faite sur la QBF est également réalisée sur l'ensemble SBP. Si un littéral ℓ universellement quantifié est propagé grâce aux prédicats, alors la branche $\overline{\ell}$ est supposée être vraie et le solveur QBF propage ℓ pour affirmer cette hypothèse (si un modèle est trouvé avec ℓ alors la symétrie indique qu'il en existe également un avec $\overline{\ell}$). Lorsque ℓ est existentiellement quantifié, la branche $\overline{\ell}$ est considérée comme incohérente, comme il est fait dans le cadre de SAT. Les résultats expérimentaux conduits lors de cette étude ont montré que l'exploitation proposée des symétries permettait d'accroître l'efficacité des solveurs QBF. Mais surtout que de nombreuses instances QBF utilisées comme *benchmarks* par la communauté contiennent des symétries : près de 70% des instances non aléatoires de l'évaluation QBF'03 (`http://www.qbflib.org/index_eval.php`).

Ce premier travail sur les symétries pour QBF, réalisé en collaboration avec Gilles AUDEMARD et Lakhdar SAÏS, publié dans **[AMS04]**, a été étendu par Saïd JABBOUR dans sa thèse [Jab08].

Sommaire

Chapitre

4

Perspectives de recherche

NOUS présentons dans ce chapitre quelques perspectives de recherche aux travaux que nous avons réalisés. Ces perspectives s'entendent à plus ou moins long terme. Les pistes à plus court terme sont identifiables par le niveau de détails techniques fournis. En effet, ces détails représentent un bon indicateur de la maturité de ces perspectives. Néanmoins, pour la majorité d'entre elles, ces pistes sont envisagées à moyen terme.

Nous reprenons la même structuration que lors de la présentation des travaux dans les deux chapitres précédents et ajoutons de nouvelles voies sur lesquelles nous envisageons de travailler à l'avenir.

4.1 SAT

4.1.1 Hybridation

L'hybridation de solveurs est au cœur de notre contribution que cela soit sur SAT ou sur les problèmes au-delà de SAT. Nous pensons donc poursuivre les développements entamés il y a maintenant plus de dix ans sur ce sujet.

Concernant l'hybridation des méthodes de résolution pour SAT, nos dernières contributions portent sur l'intégration des techniques d'analyse de conflits et d'apprentissage de clauses au sein des méthodes de recherche locale. Les premiers développements que nous avons réalisés dans ce sens sont extrêmement encourageants puisque les méthodes hybrides développées obtiennent de bien meilleures performances que toutes les autres méthodes hybrides testées et dépassent également les approches classiques de recherche locale. Néanmoins, les méthodes jusqu'alors développées n'atteignent pas les performances des meilleures approches CDCL. Cette différence de performance mesure tout l'effort qu'il reste à fournir sur le développement de nouvelles techniques hybrides. Les pistes que nous envisageons d'explorer dans l'avenir sont multiples sur ce domaine.

La construction du graphe d'implications dans les méthodes de recherche locale, telle que nous l'avons proposée dans **[ALMS09b]** passe par une étape de reconstruction explicite d'un arbre de recherche à la DPLL. Avec Gilles AUDEMARD, Jean-Marie LAGNIEZ et Lakhdar SAÏS, nous cherchons à éviter la reconstruction de cette arborescence. Pour ce faire, nous envisageons d'exploiter les notions de clauses critiques et liées afin de construire un chemin de clauses critiques à partir duquel une nouvelle clause pourra être produite par résolution et le littéral à flipper déterminé.

Une seconde piste sur laquelle nous envisageons de travailler est l'exploitation au sein d'un solveur hybride SAT des récents progrès faits sur le calcul de MUS. En effet, les algorithmes de calcul de MUS que nous avons développés reposent sur une exploitation forte de la recherche locale afin d'identifier les clauses susceptibles d'appartenir à un MUS. Il s'avère que sur certaines instances nos algorithmes vont plus vite à délivrer une approximation d'un MUS que les meilleures approches CDCL à prouver que l'instance dans sa globalité n'admet pas de modèle. Ce résultat renforce nos convictions sur la qualité de l'information que peut délivrer une méthode de recherche locale pour la résolution d'instances non satisfiables. Nous envisageons de développer une technique hybride où des sous-ensembles de clauses délivrés par la méthode de recherche locale sont testés par une approche CDCL. Cette technique proche de AOMUS différera par la manière de construire les sous-ensembles de clauses testés et par une plus forte coopération des deux solveurs avec notamment la transmission de clauses apprises.

On constate aujourd'hui une explosion du nombre de cœurs dans les processeurs. Il est évident que les futurs solveurs devront exploiter l'ensemble de la puissance de calcul offerte par ces processeurs. Nous revenons dans la section 4.1.5 sur l'adaptation des solveurs à ces nouvelles architectures. Cependant, il est clair que ces avancées technologiques incitent à utiliser et à faire coopérer différentes méthodes. Nous pensons donc développer des algorithmes hybrides dédiés à ces architectures « multicore ».

4.1.2 Méthode de recherche locale

L'un de nos premiers travaux de recherche portait sur les méthodes de recherche locale pour SAT et plus précisément sur la mise-en-œuvre d'un algorithme tabou pour SAT **[MSG97c]**. Sur ce type particulier d'algorithmes, qui se retrouve au cœur d'un bon nombre de nos autres recherches (hybridation, calcul de noyaux, etc.), nous envisageons d'explorer deux nouvelles pistes.

La première piste consiste en l'adaptation à SAT du concept de recherche à voisinage variable [MH97, HM09]. Le voisinage dans les méthodes de recherche locale pour SAT est réduit aux interprétations qui diffèrent de l'interprétation courante par un seul littéral. Ce littéral appartient à au moins une clause falsifiée par l'interprétation courante. Le voisinage pour SAT est donc réduit à un sous-ensemble des interprétations distantes de 1, en terme de distance de HAMMING. L'idée des méthodes à voisinage variable est d'étendre ce voisinage, en explorant les interprétations distantes de 2, 3, ... jusqu'à trouver une interprétation voisine qui améliore la configuration courante en terme du nombre de clauses falsifiées. Afin de contrôler l'explosion combinatoire des interprétations voisines, nous envisageons, une fois de plus, d'exploiter les concepts de clauses critiques et liées. Le voisinage de 2 sera calculé parmi les interprétations qui diffèrent de l'interprétation courante par deux littéraux appartenant respectivement à une clause critique et à une clause liée à cette clause critique. Puis les clauses liées à cette dernière clause seront explorées pour construire le voisinage de distance 3, le processus se répétant jusqu'à trouver une meilleure interprétation ou qu'un cycle apparaisse dans les littéraux choisis. Dans ce dernier cas, différentes stratégies peuvent être étudiées : revenir en arrière sur un choix de littéral, sur le premier littéral ou effectuer une analyse de conflits afin d'apprendre une clause. Cette étude est en cours de réalisation avec Jean-Marie LAGNIEZ.

Toujours concernant l'amélioration des méthodes de recherche locale pour SAT, nous explorons une seconde manière d'étendre le voisinage. Au lieu d'explorer un voisinage qui va chercher à satisfaire au moins une clause parmi les clauses falsifiées, nous proposons d'explorer les interprétations voisines

telles que toutes les clauses critiques et liées sont satisfaites. Cette méthode qui peut s'apparenter à la méthode d'inversion proposée par Thierry CASTELL dans sa thèse [Cas97] où lorsqu'un minimum local est atteint, l'algorithme flippe l'ensemble des variables ce qui a pour effet de « sur-satisfaire » toutes les clauses falsifiées. Notre objectif est de déterminer l'interprétation qui satisfait obligatoirement l'ensemble de ces clauses fausses et qui falsifie le moins de clauses au total. Pour ce faire, nous pensons exploiter certaines propriétés des clauses critiques et liées dans un minimum local.

4.1.3 Exploitation des propriétés structurelles

Concernant l'exploitation de propriétés structurelles, jusqu'alors nos travaux ont concerné les approches complètes de type DPLL. Nous envisageons une exploitation de ces propriétés et notamment des portes booléennes dans des algorithmes de recherche locale. Dans [PTS07][30], les auteurs ont utilisé l'approche que nous avions proposée dans **[OGMS02]** afin d'utiliser les dépendances fonctionnelles de la forme $y = f(x_1, x_2, ..., x_n)$ avec $f \in \{\Leftrightarrow, \vee, \wedge\}$ au sein d'un algorithme de recherche locale. Ces dépendances servent à scinder en deux l'ensemble des variables. Cette approche applique une restriction du processus de recherche locale à un ensemble de variables indépendantes obtenu à partir des portes extraites de la formule CNF. La comparaison expérimentale réalisée de leur approche concerne une catégorie d'instances (*parity*, *ssa*) qui n'encodent pas de problème réel. Les seules instances issues d'applications réelles considérées dans le papier sont les instances *bart* pour lesquelles l'approche proposée est mille fois plus lente qu'une approche classique de recherche locale, alors que l'objectif était de rendre les approches de recherche locale plus efficaces sur les instances structurées. Nos derniers résultats sur l'extraction de portes **[GMOS05]**, combinés à la technique proposée dans [AS07a] au sujet des transformations polynomiales de formules CNF sous forme de circuit combinatoire ouvrent de nouveaux horizons pour une exploitation effective des dépendances fonctionnelles au sein d'un algorithme de recherche locale. En effet, ces approches permettent l'obtention de dépendances fonctionnelles même si la formule CNF ne contient pas syntaxiquement ce type de structures. Ceci étend sensiblement notre précédente approche syntaxique à la base de la technique récompensée à *IJCAI'07*. Cette différence fondamentale nous permet d'envisager un traitement plus large d'instances SAT structurées. Deux catégories de variables pouvant être dérivées des portes seront également considérées : les variables dépendantes dont l'exploitation équivaut à la manipulation de sous-formules et les variables de l'ensemble coupe-cycle dont l'assignation conduit à un graphe associé acyclique.

Une autre voie pour étendre nos travaux sur la recherche de portes concerne la production de nouvelles clauses permettant de détecter par un moyen syntaxique ou sémantique des portes jusqu'alors non identifiées. Étant donné un ensemble de clauses, nous envisageons d'enrichir cet ensemble avec de nouvelles clauses particulières. Ces nouvelles clauses devront être déduites de préférence par un processus polynomial comme par exemple la déduction logique réduite à la propagation unitaire. L'objectif est en quelque sorte d'essayer de compléter une porte afin de la détecter plus facilement. Une partie de ce travail est en cours d'étude avec la collaboration de Cédric PIETTE et de Lakhdar SAÏS.

Par ailleurs, une fois la formule réécrite sous forme d'un ensemble de portes booléennes, celle-ci peut être assimilée à un circuit logique. Or de nombreux travaux existent dans le domaine particulier de la conception et la vérification de circuits [Puc90, Che06] et particulièrement de nombreuses études ont été conduites sur des techniques de simplifications de circuits (voir par exemple [KS02]). Il nous semble intéressant de mettre en rapport ces travaux avec les nôtres afin d'exploiter de nouvelles propriétés structurelles combinant les opérateurs $\Leftrightarrow$, $\wedge$ et $\vee$.

L'utilisation d'un formalisme étendu à base de portes booléennes ouvre un problème plus général qui est la recherche d'un bon compromis entre la représentation d'un problème et l'efficacité des algo-

[30] Ce papier a reçu un « *Best paper award* » lors de la conférence *IJCAI'07*.

rithmes associés. Ce problème est bien connu mais très actuel dans le cadre SAT. En effet, le codage de problèmes en CNF reste aujourd'hui le modèle le plus utilisé, car il permet d'allier la simplicité de la représentation et l'efficacité des algorithmes. Il reste que ce codage peut dans certains cas donner lieu à des formules de tailles extrêmement importantes. Certaines instances dépassent la capacité de la mémoire disponible et même dans le cas où l'instance peut être mémorisée, le temps de lecture peut se révéler aussi prohibitif. L'obstacle n'est donc pas la difficulté éventuelle de l'instance mais juste sa taille ! Ce problème de représentation constitue un sujet d'étude que nous souhaitons développer.

Même si la représentation sous forme clausale est conservée, le codage de problèmes est une étape indispensable à la résolution efficace de celui-ci. Souvent, la manière de modéliser un problème (par exemple, déterminer quels objets représenter et quels types de relations entre ces objets) influe considérablement sur la difficulté de résolution du problème. Dans le cadre des applications, il est important de considérer à la fois les améliorations au niveau de l'algorithmique et au niveau du codage. Un bon codage est celui qui favorise une résolution efficace. On sait par exemple que la notion de clauses redondantes a une influence sur l'efficacité des algorithmes de résolution. Cet aspect de modélisation des problèmes en CNF n'a que peu ou pas été développé dans nos activités de recherche passées, mais est un point particulièrement intéressant sur lequel nous désirons apporter une contribution future.

4.1.4 Techniques de simplification

La taille des problèmes résolus par les solveurs SAT modernes (jusqu'à plusieurs millions de variables et de clauses) démontre clairement que le caractère NP-complet de ce problème demeure essentiellement un verrou théorique, mais ne constitue pas un obstacle à la mise en œuvre de techniques algorithmiques le plus souvent efficaces en pratique. Au-delà de cette efficacité pratique, d'autres instances demeurent hors de portée de toutes les techniques de résolution dont nous disposons à l'heure actuelle. On peut alors se demander comment caractériser la sous-formule représentative de la difficulté réelle d'une instance de SAT. La mise en œuvre de techniques de simplification ou de pré-traitements polynomiaux capables de réduire sensiblement la taille des instances SAT peut être vue d'une certaine manière comme un moyen permettant d'approcher la sous-formule représentative de la difficulté réelle de l'instance. Dans la continuité de nos travaux antérieurs concernant la mise en œuvre de techniques de simplification efficaces, notre étude s'articulera selon deux parties complémentaires.

Dans la première partie, nous souhaitons développer des techniques capables de réduire en premier lieu la partie non traitable de la formule (n'appartenant pas à une classe polynomiale donnée), en définissant des opérateurs de réduction efficaces. L'objectif est de fournir une approche, de préférence polynomiale, permettant de plonger la formule dans un fragment traitable cible. Un premier pas dans ce sens a été réalisé avec, par exemple, la suppression de clauses u-redondantes mais d'autres techniques de simplification exploitant la structure des formules sont également à considérer.

Dans la seconde partie, une voie que nous souhaitons explorer est issue de la constatation suivante : éliminer des variables, des clauses, ou encore en produire de nouvelles peut amener sur certaines classes d'instances à des formules encore plus difficiles à résoudre ! Nous souhaitons proposer différents types de mesure de pertinence d'une variable ou d'une clause relativement à la formule pour maximiser l'apport de son élimination ou de son ajout en terme d'efficacité. Cette étude ambitieuse peut conduire au développement de techniques capables d'éliminer des clauses « superflues » ou redondantes et de les remplacer par des clauses plus « pertinentes ».

4.1.5 Multicore

Les architectures multi-cœurs sont la réponse offerte par les sociétés de conception de processeurs pour accroître les performances des nouveaux ordinateurs. En effet, dans l'état actuel des connaissances,

la course au « *Ghz* » a atteint ses limites et, dans le but d'offrir des performances toujours supérieures, les « fondeurs » ont multiplié le nombre d'unités de calcul au sein d'un même processeur, permettant ainsi à fréquences équivalentes d'effectuer plusieurs calculs à la fois. Les processeurs dits « *dual core* » sont aujourd'hui les plus répandus, les « *quad core* » représentent près du tiers des nouvelles ventes et les processeurs de nouvelles générations intégreront 6, 8, 16, 32, ... 1024 cœurs. Le problème relaté par beaucoup d'experts est que les fabricants de processeurs font évoluer leurs puces multi-cœurs trop vite et que l'industrie du logiciel a du mal à suivre. Afin de ne pas aggraver ce constat, il est important que les logiciels développés dans le cadre de la recherche intègrent directement ces évolutions. L'adaptation des programmes aux architectures multi-cœurs est d'autant plus vitale lorsqu'il s'agit de traiter des applications qui ont besoin d'une quantité énorme de ressources de calcul comme la résolution d'instances de SAT.

La recherche sur l'adaptation logicielle des solveurs SAT a fait l'objet de quelques études récentes [SLB05, SV06, LSB07, SM08, HJS08, HJS09b, HJS09a, OU09] mais de nombreux progrès restent à accomplir. En particulier, le solveur appelé *manysat* a permis de montrer que l'utilisation d'approches coopératives et orthogonales est nettement plus appropriée dans le contexte de solveurs SAT modernes sur architecture multi-cœurs. *manysat* profite de la principale faiblesse des approches CDCL : leurs sensibilités aux paramètres de réglage. Par exemple, l'évolution des paramètres liés à la stratégie de redémarrage ou à l'heuristique de choix de variable peut changer complètement l'exécution d'une résolution sur un problème particulier. Dans un contexte multi-cœur, ce solveur profite de ce manque de robustesse pour exécuter différentes instances d'un solveur sur un problème particulier. Chaque instanciation du solveur devra exploiter un ensemble de paramètres différents et leur combinaison devra représenter un ensemble de stratégies orthogonales. Les éléments suivants peuvent être utilisés pour différencier chacune des stratégies :

- la stratégie de choix de variables ;
- la stratégie de choix de valeurs ;
- la stratégie de redémarrage avec notamment de nouvelles politiques dynamiques ;
- etc.

Afin de permettre au solveur de retenir la meilleure stratégie possible, le partage des clauses apprises a été introduit dans *manysat* et s'avère être fondamental dans la réalisation de nouveaux solveurs multi-programmés. Ce partage d'information doit faire l'objet d'études approfondies tant sur la nature de l'information à partager entre les cœurs que sur la manière de partager efficacement cette information entre les cœurs (utilisation de mémoire cache partagée, envoi asynchrone, ...). Clairement, les clauses partagées et les différentes stratégies utilisées interagissent. Un objectif de ce travail consiste donc à faire des avancées à la fois théoriques et pratiques sur ces questions fondamentales.

Un autre point autour de *manysat* concerne la capacité de ce solveur à utiliser un plus grand nombre de cœurs. Pour l'instant ce solveur a été essentiellement testé sur des architectures *quad core* et le paramétrage de chaque instanciation de solveur est réalisé manuellement. Il serait intéressant de régler automatiquement ces paramètres et ceci pour un nombre de cœurs plus important et pouvant même varier au cours de la résolution. L'objectif est d'obtenir un solveur pour lequel « *l'extensibilité* »[31] et la « *reconfiguration automatique* »[32] soient réelles.

Une seconde manière d'envisager l'utilisation de plusieurs unités de calcul simultanément est la réalisation de solveurs type « *portfolio* ». Les solveurs SAT sont capables de traiter des problèmes avec des centaines de milliers de variables ou plus. Toutefois, il semble que chaque catégorie de solveur possède ses instances privilégiées. Les approches de type CDCL se montrent particulièrement performantes sur les benchmarks industriels, alors que les approches de type recherche locale stochastique ou de type

[31] « *scalability* » en anglais.
[32] « *reconfigurable computing* » en anglais.

« *look-ahead* » (comme LSAT **[OMS03]** ou *kcnfs* [DD04]) le sont sur des instances aléatoires et « *artisanales* »[33]. Notre objectif est donc de concevoir un solveur multi-cœur qui comprend un portfolio de différents types de techniques (« *lookback* », « *lookahead* », recherche locale, filtrages, etc.). Le défi est de déterminer le meilleur portfolio pour une instance donnée, et d'intégrer dynamiquement les techniques retenues dans un solveur multi-programmé. Ce travail peut être vu, au moins dans une certaine mesure, comme une extension de l'approche introduite dans *satzilla* [XHHLB08].

4.2 Au-delà de SAT

4.2.1 Calcul de noyaux : cadre propositionnel

Dans le cadre de la recherche de noyaux incohérents pour SAT, nous avons développé de nouveaux algorithmes, souvent basés sur l'hybridation de techniques complètes et de recherche locale stochastique, permettant d'approximer ou de calculer exactement un MUS, l'ensemble des MUS d'une formule ou un sous-ensemble des MUS caractérisant toutes les zones disjointes d'incohérence. Ces travaux offrent de nombreuses perspectives.

Une grande partie des techniques mises au point utilisent de nouvelles heuristiques dédiées à l'extraction d'ensembles de contraintes minimalement inconsistants. Si les notions mises en jeu ont été étudiées précisément, certaines d'entre elles peuvent sans doute être étendues ou généralisées. Par exemple, le concept de clause critique permet de prendre en considération un voisinage partiel des interprétations parcourues par une recherche locale. Il peut sembler intéressant d'étudier l'utilité d'un voisinage plus grand pour la capture de MUS.

Ces différents algorithmes qui se révèlent très efficaces en pratique comparativement aux techniques existantes nous permettent d'envisager leur exploitation et leur extension à d'autres fins. Notamment, le problème du calcul du plus petit MUS (SMUS « *Smallest Minimally Unsatisfiable Subformula* »), parfois nécessaire dans certaines applications, n'est possible aujourd'hui que pour des instances de taille raisonnable. Un premier objectif concerne le développement des techniques permettant d'augmenter de manière significative les performances des approches actuelles reposant essentiellement sur le principe du « *branch and bound* ».

Enfin, le calcul des MUS, l'obtention du nombre maximal de clauses qu'il est possible de satisfaire (MAXSAT) ou encore le calcul de modèles préférés ou minimaux sont des problèmes fortement connexes. Un objectif réside donc dans la proposition d'algorithmes exploitant les derniers travaux sur les MUS pour la résolution pratique de ces problèmes.

4.2.2 Calcul de noyaux : cadre CSP

Le concept de MUST introduit dans nos travaux sur la localisation de l'incohérence dans des CSP offre un cadre propice à de nombreux travaux futurs. Une première voie de recherche sur laquelle nous désirons travailler est la détection de MUST directement sur l'ensemble du CSP et non sur un MUC de celui-ci. Nous avons vu qu'un CSP pouvait contenir des MUST qui n'appartenaient à aucun MUC du CSP. Il s'agira dans un premier temps de caractériser ces ensembles particuliers de tuples interdits afin d'éviter leur calcul ou au contraire de les exploiter à d'autres fins. En effet, il pourrait également être intéressant d'étudier dans quelle mesure les MUST peuvent offrir un nouveau schéma de filtrage de CSP ou permettre de raffiner les algorithmes de filtrage existants.

Une autre piste réside dans l'adaptation du concept de couvertures incohérentes à la fois au niveau des MUC mais également au niveau des MUST. L'étude de la pertinence de ces ensembles pour la

[33] « *crafted* » en anglais.

restauration de la cohérence constitue un sujet d'études intéressant sur lequel nous désirons travailler. Des premiers travaux dans ce sens ont été réalisés en collaboration avec Éric GRÉGOIRE et Cédric PIETTE.

4.2.3 Compilation des bases de connaissances

Nos travaux sur la compilation de bases de connaissances reposent principalement sur la procédure DPLL. Nous pensons qu'une voie naturelle pour améliorer les performances des algorithmes proposés réside dans l'utilisation d'heuristiques de branchement spécialisées. Par exemple, lors du calcul d'une couverture de simplifications traitables, il serait intéressant de déterminer dans quelle mesure il est possible de mettre en œuvre une heuristique favorisant l'apparition de formules traitables. Par ailleurs, seul un nombre restreint de classes traitables a été considéré dans les algorithmes proposés. Une autre amélioration possible serait d'exploiter un plus large panel de classes traitables.

4.3 Applications

Les termes « *applications* » et « *problèmes réels* » ont très souvent été utilisés au cours de cette synthèse. Outre l'intérêt d'utiliser les instances issues d'applications réelles comme base de tests pour l'évaluation pratique des techniques de résolution développées, il me paraît important de considérer les applications depuis la phase de modélisation jusqu'à la résolution en passant par l'étape de codage. En plus du lien étroit entre ces différentes phases, cette approche permet souvent d'exhiber de nouvelles directions de recherche. En effet, les applications permettent de montrer non seulement les limites des méthodes de résolution, mais également les difficultés de représentation. Cela montre l'importance d'une telle perspective. Je reste cependant conscient des difficultés et du travail nécessaire pour faire un pas significatif dans cette direction. Néanmoins, j'envisage une telle démarche pour mes travaux futurs.

Partie II

Curriculum vitae

Sommaire

Notice Individuelle

État civil

Bertrand Mazure
Né le 30 janvier 1973 à Hénin-Beaumont (Pas-de-Calais, France)
Nationalité française, marié, un enfant.

Fonction actuelle

Maître de Conférences Classe Normale (section CNU 27 informatique)
Lieu d'exercice : UFR des Sciences, Université d'Artois, Lens.
Laboratoire de rattachement : **C**entre de **R**echerche en **I**nformatique de **L**ens (UMR8188)

Coordonnées professionnelles

CRIL-CNRS UMR8188
Université d'Artois
UFR des Sciences JEAN PERRIN, bâtiment B, bureau P301
rue Jean Souvraz, SP18
F-62307 Lens
+33 (0)3 21 79 17 87
+33 (0)3 21 79 17 70
bertrand.mazure@cril.fr
http://www.cril.fr/~mazure

Parcours professionnel

1999 – . . .	**Maître de Conférences**, UFR des Sciences, Université d'Artois, Lens. Bénéficiaire d'une **PEDR** depuis octobre 2001 (renouvelée en octobre 2005)
1998 – 1999	**ATER** à temps complet, UFR des Sciences, Université d'Artois, Lens.
1995 – 1998	**Allocataire de Recherche** (MESRT), UFR des Sciences, Université d'Artois, Lens.
	Moniteur de l'enseignement supérieur
1995 – 1995	**Vacataire** chargé d'enseignement (3 mois), UFR des Sciences, Université d'Artois, Lens.

Cursus universitaire

1999	**Doctorat en Informatique**, Université d'Artois, co-encadré par Éric GRÉGOIRE et Lakhdar SAÏS « De la Satisfaisabilité à la Compilation de Bases de Connaissances Propositionnelles », Mention : Très Honorable.
1995	**DEA d'Informatique**, Université de Lille I, Laboratoire d'accueil : CRIL, Mention : Bien.
1994	**Maitrise d'Informatique**, Université de Lille I, Mention : Assez-Bien.
1993	**Licence d'Informatique**, Université de Lille I, Mention : Assez-Bien.
1992	**DEUG Sciences et Structures de la Matière**, Université de Lille I, Spécialité Mathématiques.

Sommaire

Activités liées à la recherche

Thèmes de recherche

L'un des axes majeurs de recherche en Informatique et plus précisément en Intelligence Artificielle concerne la représentation des connaissances et le raisonnement à partir de ces connaissances. Une approche « classique » repose sur les concepts de la logique mathématique. Le problème est qu'en informatique, on attend des résultats pratiques : la déduction doit être programmée et produire ces résultats en temps raisonnable. C'est dans ce cadre très général que s'inscrivent mes travaux de recherche.

En fait, de nombreux problèmes pratiques peuvent se représenter de manière simple en logique propositionnelle. Cependant il reste que le SAT[34] est NP-complet. La déduction en calcul propositionnel n'admet pas, à l'heure actuelle, d'algorithmes généraux et efficaces dans tous les cas. Tant que la conjecture $P = NP$ n'a pas été prouvée ou réfutée, on ne saura pas s'il existe des algorithmes efficaces, c'est-à-dire de complexité en temps au pire polynomiale, pour résoudre ce problème ainsi que ceux qui s'y rapportent naturellement ou qui le contiennent (démonstration automatique, programmation logique, bases de données déductives, vérification de circuits, ...).

L'objectif principal de mon travail porte sur l'élargissement de la classe des problèmes traitables aux moyens de nouvelles techniques, heuristiques ou algorithmes permettant d'accélérer en pratique les traitements, ou encore par la découverte de restrictions pour lesquelles on peut garantir une résolution polynomiale en temps.

Mes travaux de recherche s'articulent autour des thèmes suivants :

- représentation des connaissances et démonstration automatique ;

[34] Problème de décision portant sur la satisfiabilité d'un ensemble de contraintes construites sur un ensemble fini de symboles propositionnels et exprimées sous forme clausale.

- résolution pratique de problèmes NP-complets ;
- logique propositionnelle : modèles et algorithmes ;
- SAT (satisfiabilité d'une formule booléenne mise sous forme normale conjonctive) ;
- formules booléenne quantifiées (QBF) ;
- problèmes de satisfaction de contraintes (CSP) ;
- compilation de bases de connaissances ;
- localisation et traitement de l'incohérence ;
- logiques non monotones ;
- travail coopératif et fusion des connaissances ;
- validation des systèmes à bases de connaissances.

J'effectue ma recherche au sein de l'axe « *Algorithmique pour l'inférence et la prise de décision* » du CRIL (Centre de Recherche en Informatique de Lens, UMR 8188) et je suis titulaire d'une PEDR (Prime d'Encadrement Doctoral et de Recherche) depuis octobre 2001 (renouvelée en octobre 2005).

Activités d'évaluation de la recherche

- **Membre du Comité National** de la Recherche Scientifique (section 07) depuis 2008
- **Membre extérieur de la CSE** (section CNU 27) de l'Université de Picardie de 2004 à 2008
- **Membre de la CSE** (section 25-26-27) de l'Université d'Artois de 2001 à 2007
- **Membre du comité de sélection** (section 27) de l'Université d'Artois depuis 2009
- **Membre du comité de programme**
  - des conférences internationales IEEE-IRI (*The IEEE International Conference on Information Reuse and Integration*) depuis 2004 et ICTAI (*International Conference on Tools with Artificial Intelligence*) depuis 2008
  - du workshop international « *Advances in Propositional Deduction* » lors de la conférence ECAI'96
  - des conférences nationales JNPC'97 (*Journées Nationales sur la résolution Pratique de problèmes NP-complets*) et RJCIA'00 (*Rencontres nationales des Jeunes Chercheurs en Intelligence Artificielle*)

- **Rapporteur** en 2003 pour le « *Research Grants Council* » de Hong-Kong
- **Relecteur** pour
  - les revues internationales *Information Sciences* et *Journal of Artificial Intelligence Research*
  - pour les conférences internationales IJCAI'03, IJCAI'07, ICTAI'03, ICTAI'04, ECAI'06, MICAI'06, SAT'05, SAT'06 et SAT'07
  - pour les conférences nationales JNPC'03, RFIA'04 et COSI'07
- **Examinateur** aux thèses de Olivier FOURDRINOY (Université d'Artois), Cédric PIETTE (Université d'Artois), Richard OSTROWSKI (Université Artois), David ANSART (Université d'Artois) et Bernard JURKOWIAK (Université de Picardie)

Participation à des projets de Recherche

2009 – ...	Projet de type BQR (Bonus Qualité Recherche) de l'Université d'Artois : « *Utilisation de la programmation par contraintes multi-cœurs pour représenter et manipuler des préférences* » [**porteur** du projet]
2009 – ...	ANR Blanc UNLOC « *Local Search and Unsatisfiability* »
2006 – ...	Membre du GDR I^3-IAF (Thème n°1 *Intelligence Artificielle Fondamentale* du Groupement De Recherche (*Information - Interaction - Intelligence*)
2006 – 2008	Projet du programme PESSOA (partenariat franco-portugais Hubert CURIEN), *« MUSICA : Algorithmes pour l'identification et le calcul de sous formules minimales inconsistantes »*, en partenariat avec le groupe SAT du laboratoire INESC, Lisbone, Portugal
2004 – 2008	Projet « *TIC : Traitement Intelligent des Connaissances* » de l'axe TACT (devenu TAC) du CPER Nord/Pas-de-Calais
2003 – 2004	AS STIC : « *Algorithmique et problématique expérimentale pour l'évaluation de formules booléennes quantifiées* » (AS CNRS STIC 83 du RTP 11 « *Information et Intelligence : Raisonner et Décider* »)
2002 – 2004	Action Universitaire Intégrée Luso–Françaises, projet « *OpenSAT : a platform open source for SAT* » en partenariat avec le groupe SAT du laboratoire INESC, Lisbone, Portugal
2001 – 2009	Projet « *Composants intelligents de connaissances* » du groupement d'intérêt scientifique IRCICA
1998 – 2006	Projet « *Algorithmes pour l'inférence et la satisfaction de contraintes* », groupe de travail 1.2 du groupement de recherche I^3 (*Information - Interaction - Intelligence*)
1995 – 2000	Projets Ganymède and Ganymède II of the CPER Nord/Pas-de-Calais on « *Communication Avancée et Activités Coopératives* »

1995 – 1996 Projet « *RESSAC : Aspects algorithmiques de la résolution de problèmes exprimés à l'aide de contraintes* » of the PRC-IA

Co-encadrement de thèses

2008 – . . . Jean-Marie LAGNIEZ « Recherche locale pour SAT et UNSAT » Direction : Lakhdar SAÏS, co-encadrant : Gilles AUDEMARD

2005 – 2007 Cédric PIETTE « Contributions à la détection de formules minimalement inconsistantes » Direction : Éric GRÉGOIRE

2003 – 2007 Olivier FOURDRINOY « Autour de l'hybridation des méthodes de résolution pour SAT » Direction : Éric GRÉGOIRE, co-encadrant : Lakhdar SAÏS

2001 – 2004 Richard OSTROWSKI « Reconnaissance et exploitation de propriétés structurelles pour la résolution du problème SAT » Direction : Éric GRÉGOIRE, co-encadrant : Lakhdar SAÏS

2000 – 2005 David ANSART « Utilisation et extensions de l'algorithmique pour SAT pour la résolution de différents problèmes d'intelligence artificielle » Direction : Éric GRÉGOIRE

Encadrement de stages de Master Recherche ou DEA

2008 Soufien GHINI « SAT : nouvelles formes de représentation et de résolution »

2008 Jean-Marie LAGNIEZ « Recherche locale pour SAT et UNSAT »

2005 Cédric PIETTE « Méta-heuristiques pour la détection de noyaux minimalement inconsistants »

2004 Jérome DEGAVE « Symétries et QBF »

2003 Olivier FOURDRINOY « Hybridation des méthodes de résolution pour SAT »

2001 Richard OSTROWSKI « Résolution de formules booléennes générales »

Distinction scientifique

2003 Compétition SAT'03 : solveur « LSATv2.0 » classé premier dans la catégorie d'instances « *handmade* » lors des phases qualificatives et second lors de la phase finale

Autres responsabilités liées à la recherche

- **Membre du groupe fondateur** de l'IRCICA de Lille (Institut de Recherche sur les Composants Matériels et Logiciels pour l'Information et la Communication Avancée)

- **Membre du comité d'organisation** des « *Premières Journées Françaises en Programmation par Contraintes* » (JFPC'05 fusion des congrès JNPC et JFPLC) qui se sont tenues à Lens en juin 2005
- **Membre du comité d'organisation** et responsable des publications de « *the 22st IEEE-International Conference on Tools with Artificial Intelligence* » (ICTAI 2010) qui se tiendront à Arras en octobre 2010
- **Membre du bureau** et secrétaire (jusqu'en 2009) de l'Association Française pour la Programmation par Contraintes (AFPC : `http://www.afpc-asso.org/`)

Sommaire

Activités liées à l'enseignement

Synopsis

J'exerce une activité d'enseignement depuis 1995 au travers des différentes fonctions que j'ai occupées : vacataire, moniteur de l'enseignement supérieur, ATER et maître de conférences.

L'ensemble de ces enseignements a été réalisé à l'U.F.R. des Sciences Jean PERRIN de l'Université d'Artois. Le public concerné est donc constitué d'étudiants en filière scientifique et aux spécialités et niveaux variés. Il s'étend d'étudiants de l'ancien DEUG « Sciences de la vie », aux étudiants de master recherche informatique deuxième année « Systèmes intelligents et applications ».

Depuis dix ans, ces enseignements représentent un volume horaire (équivalent TD) de près de 250 heures par an.

Synthèse des enseignements

- Bureautique, utilisation de la suite Office — DEUG SV 2e année *TP*
- Initiation à l'algorithmique — DEUG MIAS 1re année et SM 2e année *CM/TD/TP*
- Programmation Pascal — DEUG MIAS 2e année *CM/TD/TP*
- **Introduction à UNIX et à la programmation shell** — Licence Informatique 3e année *CM/TD/TP*
- Introduction à la programmation C — Licence Informatique 3e année *CM/TD/TP*

❏ Introduction à la programmation Lisp	Licence Informatique 3e année *TP*
❏ Introduction à la programmation Prolog	Licence Informatique 3e année *TP*
❏ Introduction à la programmation Java	Licence Informatique 3e année *TP*
❏ Algorithmique avancé	Licence Informatique 3e année *TD/TP*
❏ **Programmation avancée en C**	Licence Informatique 3e année *CM/TD/TP*
❏ Réseaux	Master Informatique 1re année *CM/TD/TP*
❏ **Systèmes d'exploitation centralisés**	Master Informatique 1re année *CM/TD/TP*
❏ **Systèmes d'exploitation distribués**	Master Informatique 1re année *CM/TD/TP*
❏ Introduction à la programmation Fortran	Master de Physique 1re année *CM/TD/TP*
❏ Satisfaction et optimisation de contraintes	Master Informatique Recherche 2e année *CM*
❏ **Introduction à la programmation Perl**	Master Informatique Professionnel 2e année *CM/TD/TP*
❏ **Administration des systèmes d'exploitation et des réseaux**	Master Informatique Professionnel 2e année *CM/TD/TP*

Les intitulés en gras, représentent les cours dont j'ai la responsabilité actuellement.

Encadrement de stages

En plus des encadrements de thèse et de master recherche détaillés précédemment (voir page 66), j'ai encadré des stages de TER au niveau master informatique 1re année. Les sujets de ces stages portaient sur :

- Programmation de différents algorithmes pour SAT
- Algorithmes pour la détection de symétries pour SAT
- Réalisation d'une application distribuée pour l'analyse et le stockage d'informations provenant d'expérimentations d'isoélectrofocalisation
- Étude de l'efficacité de la k-résolution sur les benchmarks modernes
- La recherche à voisinage variable dans les algorithmes de recherche locale pour SAT

Autres responsabilités liées à l'enseignement

- Correspondant « Apogée » pour la filière Informatique de l'UFR des Sciences de l'Université d'Artois, jusqu'en 2007
- Membre de la commission de validation d'études de l'UFR des Sciences de l'Université d'Artois pour la filière Informatique
  Président de cette commission pour le master Informatique 1re année
- Gestion et administration du matériel de travaux pratiques des deuxième et troisième cycles informatique de l'UFR des Sciences de l'Université d'Artois
- Participation annuelle aux journées portes ouvertes et aux journées « Fête de la Science » organisées à l'UFR des Sciences de l'Université d'Artois

Sommaire

Activités administratives

Fonctions électives

2008 – . . .	Membre élu de la section 07 du Comité National de la Recherche Scientifique
2007 – . . .	Membre élu du conseil d'UFR de la Faculté des Sciences de l'Université d'Artois
2006 – . . .	Membre élu du conseil du bureau et secrétaire (jusqu'en 2009) de l'Association Française pour la Programmation par Contraintes (AFPC : `http://www.afpc-asso.org/`)
2009 – . . .	Membre élu du Comité de Sélection (section CNU 27) de l'Université d'Artois
2001 – 2007	Membre élu de la CSE (sections CNU 25-26-27) de l'Université d'Artois
2000 – . . .	Membre élu du conseil de laboratoire du CRIL

Autres Responsabilités

- Membre de la commission budgétaire de l'UFR des Sciences de l'Université d'Artois
- Correspondant informatique pour l'UFR des Sciences auprès de la division informatique de l'Université d'Artois
- Administrateur des serveurs informatiques du CRIL situés à la Faculté des Sciences de Lens
- Administrateur du cluster de calcul (608 unités de calcul) du CRIL sur lequel sont organisées les compétitions internationales sur SAT, Pseudo-booléen et CSP
- Responsable de la cellule informatique du CRIL en charge :
  - du parc informatique du CRIL (faculté des sciences)
  - du cluster de calcul
  - et des achats informatiques du CRIL

Sommaire

Liste des Publications

Synthèse

- **6** articles parus dans des revues d'audience internationale avec comité de rédaction (EJOR, IJAIT, Information Sciences, Annals of Math/AI, IJCIS, Constraints)
- **2** articles publiés dans des revues d'audience nationale avec comité de rédaction (revues I3 et RIA)
- **1** chapitre dans le livre « Problème SAT : progrès et défis »
- **29** communications à des manifestations d'audience internationale avec actes et comité de lecture (dont IJCAI, ECAI, AAAI, CP, ICTAI, SAT, ...)
- **16** communications à des manifestations d'audience nationale avec actes et comité de lecture (dont RFIA, JFPC, JNPC, ...)
- **6** communications à des manifestations avec comité de lecture, sans acte ou avec actes électroniques

Revues Internationales avec comité de rédaction

[1] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Using local search to find MSSes and MUSes »,
European Journal of Operational Research , vol. 199, n°3, pages 640-646, décembre 2009.

[2] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« On Finding Minimally Unsatisfiable Cores of CSPs »,
International Journal on Artificial Intelligence Tools (IJAIT), vol. 17, n°4, pages 745 - 763, août 2008.

[3] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Local-Search Extraction of MUSes »,
Constraints , vol. 12, n°3, pages 325-344, septembre 2007.

[4] Éric GRÉGOIRE, **Bertrand MAZURE**,
« About the incremental validation of first-order stratified knowledge-based decision-support systems »,
Information Sciences (IS), vol. 142, Elsevier, pages 117-129, 2002.

[5] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Boosting complete techniques thanks to local search »,
Annals of Mathematics and Artificial Intelligence (AMAI), vol. 22, pages 309-322, 1998.

[6] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« An efficient technique to ensure the logical consistency of cooperative agents »,
International Journal of Cooperative Information Systems (IJCIS), vol. 6, n°1, pages 27-36, 1997.

Revues Nationales avec comité de rédaction

[7] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Extraction d'ensembles minimaux conflictuels basée sur la recherche locale »,
Revue d'Intelligence Artificielle (RSTI- RIA), vol. 22, n°2, pages 161-181, avril 2008.

[8] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« MUST et couvertures de MUST pour l'explication et la réparation de CSP incohérents au niveau »,
Information-Interaction-Intelligence (Revue I3), vol. 8, n°2, pages 181-202, 2008.

Chapitre d'ouvrage

[9] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Sous-formules minimales insatisfaisables »,
chapitre 8 de « Problème SAT : progrès et défis », Lakhdar SAÏS éditeur, Hermes, 2008.

Conférences d'audience internationale avec comité de lecture et actes

[10] Gilles AUDEMARD, Jean-marie LAGNIEZ, **Bertrand MAZURE**, Lakhdar SAÏS,
« Learning *local search* »,
dans *21st International Conference on Tools with Artificial Intelligence (ICTAI'09)*, IEEE Computer Society, novembre 2009 (à paraître).

[11] Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Local autarkies searching for the dynamic partition of CNF formulae »,
21st International Conference on Tools with Artificial Intelligence (ICTAI'09), IEEE Computer Society, pages to appear, novembre 2009.

[12] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Does this set of clauses overlap with at least one MUS ? »,
22nd International Conference on Automated Deduction (CADE 22), LNCS 5663, pages 100-115, août 2009.

[13] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« On Approaches to Explaining Infeasibility of Sets of Boolean Clauses »,
The 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08), pages 74-83, novembre 2008.

[14] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« MUST : Provide a Finer-Grained Explanation of Unsatisfiability »,
13th International Conference on Principles and Practice of Constraint Programming (CP'07), LNCS 4741, pages 317-331, septembre 2007.

[15] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Reducing hard SAT instances to polynomial ones »,
2007 IEEE international conference on Information Reuse and Integration (IEEE-IRI'07), pages 18-23, août 2007.

[16] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Eliminating Redundant Clauses *Sat Instances* »,
dans *The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, LNCS 4510, Springer, pages 71-83, mai 2007.

[17] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Boosting a Complete Technique to Find MSS and MUS thanks to a Local Search Oracle »,
International Joint Conference on Artificial Intelligence (IJCAI'07), pages 2300-2305, janvier 2007.

[18] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Tracking MUSes and Strict Inconsistent Covers »,
Sixth ACM/IEEE International Conference on Formal Methods in Computer Aided Design (FMCAD'06), pages 39-46, novembre 2006.

[19] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE, Lakhdar SAÏS,
« A New Heuristic-based albeit Complete Method to Extract MUCs from Unsatisfiable CSPs »,
the IEEE International Conference on Information Reuse and Integration (IEEE-IRI'2006), pages 325-329, septembre 2006.

[20] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Extracting MUSes »,
17th European Conference on Artificial Intelligence (ECAI'06), pages 387-391, août 2006.

[21] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Exploring Hybrid Algorithms for SAT »,
12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'05), pages 33-37, décembre 2005 (papier court)

[22] Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, **Bertrand MAZURE**, Richard OSTROWSKI, Lakhdar SAÏS,
« Using Boolean Constraint Propagation for Sub-clause Deduction »,
11th International Conference on Principles and Practice of Constraint Programming (CP'05), LNCS 3709, Springer Verlag, pages 757-761, octobre 2005.

[23] Éric GRÉGOIRE, **Bertrand MAZURE**, Richard OSTROWSKI, Lakhdar SAÏS,
« Automatic extraction of functional dependencies »,
Theory and Applications of Satisfiability Testing : 7th International Conference (SAT 2004), Revised Selected Papers, (SAT'04 Revised Selected Papers), LNCS 3542, pages 122-132, 2005.

[24] Gilles AUDEMARD, **Bertrand MAZURE**, Lakhdar SAÏS,
« Dealing with Symmetries *Quantified Boolean Formulas* »,
Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04), pages 257-262, 2004.

[25] Éric GRÉGOIRE, Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS,
« Automatic extraction of functional dependencies »,
Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04), 2004.

[26] Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Eliminating redundancies *SAT search trees* »,
15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03), pages 100-104, novembre 2003.

[27] Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently »,
the Tenth International Conference on Artificial Intelligence : Methodology, Systems, Applications (AIMSA'2002), LNCS 2443, Springer Verlag, pages 51-60, septembre 2002.

[28] Richard OSTROWSKI, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Recovering and exploiting structural knowledge from CNF formulas »,
the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02), LNCS 2470, Springer Verlag, pages 185-199, septembre 2002.

[29] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« System Description : CRIL Platform for SAT »,
the 15th Intl. Conf. on Automated Deduction (CADE-15), LNCS 1421, Springer Verlag, pages 124-128, juillet 1998.

[30] Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Logically-complete local search for propositional nonmonotonic knowledge bases »,
the 7th Intl. Workshop on Nonmonotonic Reasoning (NMR'98), pages 37-45, juin 1998.

[31] Yacine BOUFKHAD, Éric GRÉGOIRE, Pierre MARQUIS, **Bertrand MAZURE**, Lakhdar SAÏS,
« Tractable cover compilations »,
15th International Joint Conference on Artificial Intelligence (IJCAI'97), pages 122-127, août 1997.

[32] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Tabu Search for SAT »,
the 14th American National Conference on Artificial Intelligence (AAAI'97), pages 281-285, juillet 1997.

[33] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Checking Several Forms of Consistency *Nonmonotonic Knowledge-Bases* »,
9th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'07), LNAI 1244, Springer Verlag, pages 122-130, juin 1997.

[34] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Local search for computing normal circumstances models »,
the International Conference on Computational Intelligence (ICCI'97), LNCS 1226, Springer Verlag, pages 55-56, avril 1997.

[35] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« A comparison of two approaches to inconsistency detecting »,
the European Symposium on Intelligent Techniques (ESIT'97), mars 1997.

[36] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« A Powerful Heuristic to Locate Inconsistent Kernels *Knowledge-Based Systems* »,
International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96), pages 1265-1269, juillet 1996.

[37] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Detecting Logical Inconsistencies »,
the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/Math'96), pages 116-121, janvier 1996.

[38] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« TWSAT : a new local search algorithm for SAT : performance and analysis »,
the Workshop of CP'95 on Solving Really Hard Problems (Wks-SRHP-CP95), pages 127-130, septembre 1995.

Conférences d'audience nationale avec comité de lecture et actes

[39] Gilles AUDEMARD, Jean-marie LAGNIEZ, **Bertrand MAZURE**, Lakhdar SAÏS,
« Analyse de conflits dans le cadre de la recherche locale »,
Journées Francophones de la Programmation par Contraintes (JFPC'09), pages 215-224, juin 2009.

[40] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Localiser des sources d'incohérence spécifiques sans les calculer toutes »,
Journées Francophones de la Programmation par Contraintes (JFPC'09), pages 95-104, juin 2009.

[41] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Explication et réparation de l'incohérence dans les CSP : de la contrainte au tuple »,
16ème congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA'08), pages 258-267, janvier 2008.

[42] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Réduction d'instances de SAT vers des instances polynomiales »,
16ème congrès francophone Reconnaissance des Formes et Intelligence Artificielle (RFIA'08), pages 388-396, janvier 2008.

[43] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Une nouvelle méthode hybride pour calculer tous les MSS et tous les MUS »,
3èmes Journées Francophones de Programmation par Contraintes (JFPC'07), pages 143-150, juin 2007.

[44] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Suppression des clauses redondantes dans des instances SAT »,
3èmes Journées Francophones de Programmation par Contraintes (JFPC'07), pages 28-37, juin 2007.

[45] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Extraction de sous-formules minimales inconsistantes »,
2nd Journées Francophones de Programmation par Contraintes (JFPC'06), pages 201-208, juin 2006.

[46] Olivier FOURDRINOY, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Heuristique d'ordonnancement des variables pour SAT »,
Congrès de la Société Française de Recherche Operationelle et d'Aide à la Décision (ROADEF'06), janvier 2006.

[47] Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, **Bertrand MAZURE**, Richard OSTROWSKI, Lakhdar SAÏS,
« Utilisation de la Propagation de Contraintes pour la Production de Sous-Clauses »,
Premières Journées Francophones de la Programmation par Contraintes (JFPC'05), pages 69-78, juin 2005.

[48] Éric GRÉGOIRE, **Bertrand MAZURE**, Richard OSTROWSKI, Lakhdar SAÏS,
« Dépendances Fonctionnelles Booléennes : Détection et Exploitation »,
Actes du Colloque sur l'Optimisation et les Systèmes d'Informations (COSI'05), pages 263-274, 2005.

[49] Éric GRÉGOIRE, Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS,
« Déduction Automatique de Dépendances fonctionnelles »,
Dixièmes Journées Nationales de la Résolution Pratique des Problèmes NP-Complets (JNPC'04), pages 171-180, juin 2004.

[50] Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Elimination des redondances dans les algorithmes de résolution de SAT »,
quatorzième Congrés Francophone AFRIF-AFIA sur la Reconnaissance des Formes et l'Intelligence Artificielle (RFIA'04), pages 1343-1350, janvier 2004.

[51] Gilles AUDEMARD, **Bertrand MAZURE**, Lakhdar SAÏS,
« Symétries et Formules Booléennes Quantifiées »,
Dixièmes Journées Nationales de la Résolution Pratique des Problèmes NP-Complets (JNPC'04), pages 43 - 53, 2004.

[52] Richard OSTROWSKI, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Techniques de simplification de CNF »,
8èmes Journées Nationales sur la Résolution Pratique des Problèmes NP-complets (JNPC'02), pages 181-194, mai 2002.

[53] Éric GRÉGOIRE, **Bertrand MAZURE**,
« Une méthode complète de recherche locale pour des bases de connaissance propositionnelles non monotones »,
Actes des Quatrièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-complets (JNPC'98), pages 91-99, 1998.

[54] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« Deux approches pour la résolution du problème SAT »,
Deuxième Conférence Nationale sur la Résolution Pratique des Problèmes NP-complets (CNPC'96), Teknéa Editions, pages 103-114, mars 1996.

Conférences avec comité de lecture, sans acte ou actes électroniques

[55] Gilles AUDEMARD, Jean-marie LAGNIEZ, **Bertrand MAZURE**, Lakhdar SAÏS,
« Integrating Conflict Driven Clause Learning to Local Search »,
International Workshop on Local Search Techniques in Constraint Satisfaction (affiliated to CP) (LSCS09), Actes électroniques, septembre 2009.

[56] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« Une méta-heuristique basée sur le comptage de contraintes falsifiées »,
First workshop on Metaheuristics (META'06), Actes électroniques, novembre 2006.

[57] Éric GRÉGOIRE, **Bertrand MAZURE**, Cédric PIETTE,
« A new local search algorithm to compute inconsistent kernels »,
6th International Meta-heuristics International Conference (MIC'05), Actes électroniques, août 2005.

[58] Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS,
« LSAT solver »,
Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02), (SAT solvers description) mai 2002.

[59] **Bertrand MAZURE**, Pierre MARQUIS,
« Theory Reasoning within Implicant Cover Compilations »,
the ECAI-96 Workshop on Advances in Propositional Deduction (Wks-APD-ECAI'96), pages 65-69, août 1996.

[60] **Bertrand MAZURE**, Lakhdar SAÏS, Éric GRÉGOIRE,
« SUN : a Multistrategy Platform for SAT »,
First International Competition and Symposium on Satisfiability Testing (SAT'96) (SAT solvers description), mars 1996.

Mémoires de thèse et de DEA

[61] **Bertrand MAZURE**,
« De la satisfaisabilité à la compilation de bases de connaissances propositionnelles »,
Thèse de Doctorat, Université d'Artois, CRIL, Lens, France, janvier 1999.

[62] **Bertrand MAZURE**,
« Expérimentations, analyse et améliorations des méthodes de résolution du problème SAT »,
Mémoire de DEA, Université de Lille 1, Lille, France, juillet 1995.

Rapports techniques

[63] Richard OSTROWSKI, **Bertrand MAZURE**, Lakhdar SAÏS,
« LSAT Solver v2.0 »,
Rapport technique, CRIL, 2003.

[64] Richard OSTROWSKI, Éric GRÉGOIRE, **Bertrand MAZURE**, Lakhdar SAÏS,
« Recovering and exploiting structural knowledge from CNF formulas »,
Rapport technique, CRIL, avril 2002.

Partie III

Sélection de publications

Synopsis

Cette partie regroupe les publications qui me semblent les plus significatives et les plus représentatives de mes travaux de recherche.

Le choix a été fait de manière à représenter les différents aspects abordés dans la synthèse (voir partie I).

- **SAT**
  - *Recherche locale* : « *Tabu Search for SAT* » **[MSG97c]**
  - *Hybridation de solveurs* :
    - « *Boosting Complete Techniques thanks to Local Search* » **[MSG98]**
    - « *Integrating Conflict Driven Clause Learning to Local Search* » **[ALMS09a]**
  - *Exploitation des propriétés structurelles* :
    - « *Recovering and exploiting structural knowledge from CNF formulas* » **[OGMS02]**
    - « *Automatic extraction of functional dependencies* » **[GMOS05]**
  - *Techniques de simplification* : « *Reducing hard SAT instances to polynomial ones* » **[FGMS07b]**
- **Au-delà de SAT**
  - *Calcul de noyaux dans le cadre propositionnel* :
    - « *Local-Search Extraction of MUSes* » **[GMP07b]**
    - « *Using local search to find MSSes and MUSes* » **[GMP09b]**
  - *Calcul de noyaux dans le cadre CSP* : « *MUST : Provide a Finer-Grained Explanation of Unsatisfiability* » **[GMP07c]**
  - *Compilation de bases de connaissance* : « *Tractable Cover Compilations* » **[BGM+97]**

Tabu Search for SAT

Article paru dans les actes de « *the Fourteenth American National Conference on Artificial Intelligence* » (AAAI'97), pages 281–285, Providence, Rhode Island, USA, juillet 2007.

Co-écrit avec Lakhdar SAÏS et Éric GRÉGOIRE.

Tabu Search for SAT *

Bertrand Mazure **Lakhdar Saïs** **Éric Grégoire**

CRIL – Université d'Artois
rue de l'Université SP 16
F-62307 Lens Cedex France
{mazure,sais,gregoire}@cril.univ-artois.fr

Abstract

In this paper, tabu search for SAT is investigated from an experimental point of view. To this end, TSAT, a basic tabu search algorithm for SAT, is introduced and compared with Selman et al. Random Walk Strategy GSAT procedure, in short RWS-GSAT. TSAT does not involve the additional stochastic process of RWS-GSAT. This should facilitate the understanding of why simple local search methods for SAT work. It is shown that the length of the tabu list plays a critical role in the performance of the algorithm. Moreover, surprising properties about the (experimental) optimal length of the tabu list are exhibited, raising interesting issues about the nature of hard random SAT problems.

Introduction

SAT, i.e. checking the satisfiability of a boolean formula in conjunctive normal form, is a canonical NP-complete problem (Cook 1971). Moreover, it is a fundamental problem in mathematical logic, automated reasoning, artificial intelligence and various computer science domains like VLSI design.

Recently, there has been a renewal of interest in understanding the nature of the difficulty of SAT (see e.g. (Chvátal & Szemerédi 1988; Dubois & Carlier 1991; Mitchell, Selman, & Levesque 1992)). At the same time, several authors have proposed new –but amazingly simple and efficient– algorithms allowing for a breakthrough in the class of computer-solvable SAT instances (see e.g. (Selman, Levesque, & Mitchell 1992; Gu 1992; Selman, Kautz, & Cohen 1993; DIMACS 1993)).

More precisely, a class of very hard SAT instances has been characterized. This class is made of random generated K-SAT instances whose probability of being satisfiable is close to 0.5 and are located at the critical point of a phase transition. These problems are most often beyond the reach of the most efficient techniques derived from conventional algorithms, like Davis and Putnam's procedure (Davis & Putnam 1960). In order to address them, two families of algorithms have been designed recently. The first one is made of logically complete techniques that aim at proving the inconsistency of SAT instances (see e.g. (Dubois *et al.* 1996; Crawford & Auton 1993)). The second one is formed of incomplete techniques based on local reparations that attempt to find a model for SAT instances. Several authors have proposed very simple local search algorithms that prove surprisingly good in solving hard large satisfiable problems (Selman, Levesque, & Mitchell 1992; Selman, Kautz, & Cohen 1993; Gent & Walsh 1993).

In this paper, tabu search for SAT is investigated from an experimental point of view. To this end, TSAT, a basic tabu search algorithm for SAT, is introduced and compared with Selman et al. Random Walk Strategy GSAT, in short RWS-GSAT (Selman, Kautz, & Cohen 1993). TSAT proves extremely competitive in the resolution of many problems, in particular hard random K-SAT instances at the critical point of the phase transition. Actually, the FTP available GSAT already contained a basic tabu option but has not been described in the literature to our best knowledge. Moreover, no investigation of the fine-tuning of its essential parameters had ever been conducted.

In the next section, the canonical K-SAT fixed-clause-length random generation model is reviewed, with emphasis on the phase transition. Then, Selman et al. RWS-GSAT algorithm is presented. TSAT is then motivated and presented. Extensive experimentations are conducted about the optimal lengths of the tabu list with respect to several criteria, leading to surprising findings. A comparison between the performance of RWS-GSAT and TSAT is then conducted. These results were first presented for a very restricted audience in 1995 (Mazure, Saïs, & Grégoire 1995). As a conclusion, some promising ideas for further research are given.

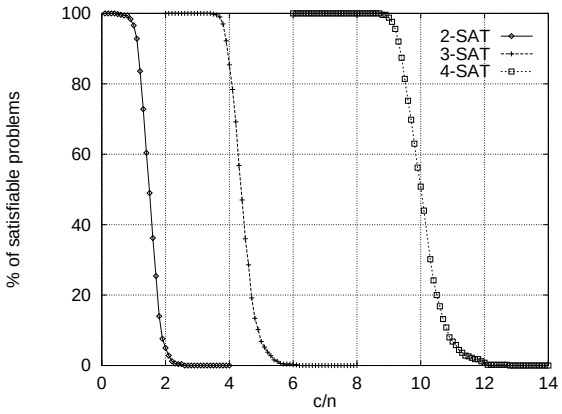


Figure 1: phase transition phenomena

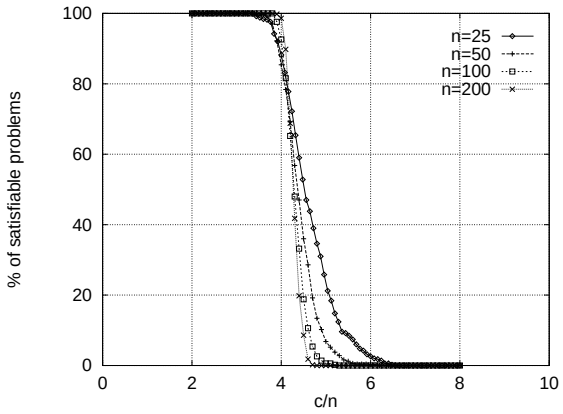


Figure 2: 3-SAT

Hard random SAT instances

SAT consists in checking the satisfiability of a boolean formula in conjunctive normal form (CNF). Let us recall here that any propositional formula can be translated into a CNF that is equivalent for SAT, thanks to a linear time algorithm (see e.g. (Siegel 1987)). A CNF formula is a set (interpreted as a conjunction) of clauses, where a clause is a disjunction of literals. A literal is a positive or a negated propositional variable. An interpretation of a boolean formula is an assignment of truth values to its variables. A model of a formula is an interpretation that satisfies the formula.

Although SAT is NP-complete, theoretical and experimental results show good average-case performance for several classes of SAT instances (see e.g. (Franco & Paull 1983)).

However, hard random instances of SAT have been observed at a phase transition. Let us illustrate this phenomenon in the usual K-SAT fixed-clause-length model (see e.g. (Chvátal & Szemerédi 1988; Cheeseman, Kanefsky, & Taylor 1991; Crawford & Auton 1993; Dubois & Carlier 1991; Mitchell, Selman, & Levesque 1992)), a random generation model where the number of literals per clause is a given value K and the sign of each literal is also randomly generated (with a 0.5 probability).

In Figures 1 and 2, we see the phase transition observed by many authors. The probability of satisfiability decreases abruptly from 1 to converge towards 0 in a phase transition as the c/n ratio increases (where c represents the number of clauses and n the number of variables). The location of the phase transition depends on the length of clauses and on the number of variables. In Figure 1, we see how the curves move to the right as this length increases. In Figure 2, we see how the curves straighten when the number of variables increases. It has been shown experimentally that instances at these phase transitions where the probability of being satisfiable is 0.5 are really hard problems. Actually, it has been proved that these problems are exponential for resolution (Chvátal & Szemerédi 1988).

Random Walk Strategy GSAT

Let us now briefly review Selman et al. GSAT algorithm (Selman, Levesque, & Mitchell 1992). This algorithm performs a greedy local search for a satisfying assignment of a set of propositional clauses. The algorithm starts with a randomly generated truth assignment. It then changes ("flips") the assignment of the variable that leads to the largest increase in the total number of satisfied clauses. Such flips are repeated until either a model is found or a preset maximum number of flips (MAX-FLIPS) is reached. This process is repeated as needed up to a maximum of MAX-TRIES times.

In the sequel, we shall consider a more recent and efficient version of GSAT, i.e. Random Walk Strategy GSAT (in short RWS-GSAT)(Selman, Kautz, & Cohen 1993). This variant of GSAT selects the variable to be flipped in the following way: it either picks with probability p a variable occurring in some unsatisfied clause or follows, with probability $1-p$, the standard GSAT scheme, i.e. makes the best possible local move.

Clearly, this very simple algorithm is logically incomplete and belongs to the local search procedures family. However, it is surprisingly efficient in demonstrating that CNF formulas are satisfiable, in particular K-SAT instances at the phase transition.

```
Procedure GSAT
Input:  a set of clauses S, MAX-FLIPS, MAX-TRIES
Output: a satisfying truth assignment of S, if found
  Begin
    for i := 1 to MAX-TRIES do
       I :=  a randomly generated truth assignment
       for j := 1 to MAX-FLIPS do
          if I  satisfies S then return I
          x :=  a propositional variable such that
                a change in its truth assignment
                gives the largest increase (possibly
                negative) in the number of clauses
                 of S  that are satisfied by I
          I := I with the assignment of x reversed
       end-for
    end-for
    return  "no satisfying assignment found"
  End
```

Figure 3: GSAT algorithm: basic version

Restricting randomness

Let us stress that the Random Walk Strategy introduces an additional level of randomness in basic GSAT and thus makes an analytical study of GSAT more difficult to conduct. Another randomness property of GSAT lies in the selection of the variable to be flipped. Indeed, as Selman et al. stress it: "Another feature of GSAT is that the variable whose assignment is to be changed is chosen *at random* from those that would give an equally good improvement. Such nondeterminism makes it very unlikely that the algorithm makes the same sequence of changes over and over (Selman, Levesque, & Mitchell 1992)".

Moving further towards the goal of avoiding recurrent flips, we investigate the use of tabu search (Glover 1989; 1990) for SAT by experimenting with an algorithm of our own, called TSAT. TSAT makes a systematic use (no aspiration criteria) of a tabu list of variables in order to avoid recurrent flips and thus escape from local minima. This technique was also expected to allow for a better and more uniform coverage of the search space. Let us stress that this use of a tabu list is systematic during the search process in the sense that the tabu list is updated each time a flip is made. TSAT keeps a fixed length –chronologically-ordered FIFO– list of flipped variables and prevents any of the variables in the list from being flipped again during a given amount of time. Accordingly, the tabu list contains *variables* and, for efficiency reasons, does not keep track of forbidden interpretations, explicitly.

The efficiency of most local search procedures depends heavily on a good setting of their parameters. For instance, Selman et al. suggest specific values for GSAT parameters like MAX-TRIES, MAX-FLIPS and, very importantly, the probability p presented above (Selman, Kautz, & Cohen 1993).

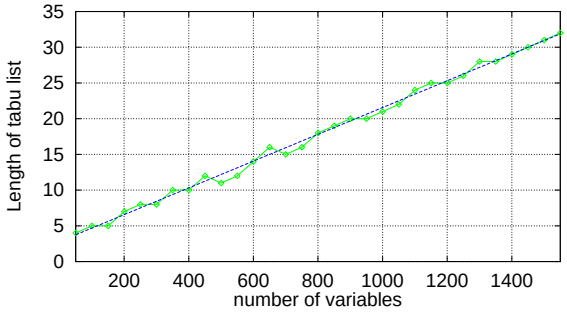


Figure 4: Optimal length of tabu lists for 3-SAT at the critical point of phase transition ($c/n = 4.25$)

In the next section, the main parameter of TSAT is fine-tuned for random K-SAT problems, namely the length of the tabu list, using the main parameters of the problem: i.e. the number of variables and the length and number of clauses.

Experimental fine-tuning of TSAT: peculiar findings

In order to find optimal lengths of the tabu list, the following extensive experimentations have been conducted. First, the 3-SAT framework has been considered. According to the standard fixed-length-clause model, 500 instances were randomly generated at the phase transition for every number of variables ranging from 50 to 1000 (by steps of 50, with their best estimated c/n ratios). The number of instances has been limited to 100 for every number of variables between 1000 and 1500 (also by steps of 50). For each such instance, TSAT has been run, varying the length of the tabu list from 1 to 50. In Figure 4, the (experimentally obtained) optimal length of the tabu list with respect to the number of variables is given. In the considered range of number of variables, this curve appears to be linear in the number of variables. Experimental result:

$$\text{optimal length of tabu list} = 0.01875 \;\; n + 2.8125$$

where n is the number of variables.

Moreover,

- a slight departure from the optimal length leads to a corresponding graceful degradation of the performance of TSAT. A more important distance from this optimal length leads to a dramatic performance degradation.
- these lengths remain optimal for random-generated instances outside the phase transition (the optimal

problems		Nb.	RWS-GSAT				TSAT			
n	c	inst.	time (sc.)	flips	solved	ratio	time (sc.)	flips	solved	ratio
100	430	500	.18	2803	88%	31.85	.11	1633	93%	17.60
200	860	500	1.99	18626	73%	255.85	.73	9678	74%	130.78
400	1700	500	15.03	204670	100%	2046.70	11.51	145710	100%	1457.10
600	2550	500	19.59	250464	62%	4013.85	13.92	167236	65%	2580.80
800	3400	500	140.61	1809986	67%	26854.39	99.45	1143444	71%	16150.34
1000	4250	500	369.88	4633763	57%	81009.84	292.10	3232463	62%	51802.29
2000	8240	50	3147.26	26542387	16%	1658899.19	3269.15	29415465	40%	735386.63

Table 1: TSAT vs. RWS-GSAT

size depends only on the number n of involved variables according to the above equation).

The above tests for 4-SAT (100 instances for each number of variables ranging from 50 to 600 (by steps of 50)) and similar values for the optimal lengths have been obtained.

TSAT vs. RWS-GSAT

Extensive experimental comparisons between RWS-GSAT and TSAT have been conducted for 3-SAT instances at the phase transition. Both algorithms have been implemented in a common platform written in C under Linux for Pentium PC, available from the authors. Our local implementation of Random Walk GSAT proves as efficient as Selman's one.

Both algorithms are best compared with respect to the percentage of solved problems and with respect to the number of performed flips. Let us stress that the tabu list is implemented as a circular list whose FIFO access is made in constant time. Time and number of flips in the Table 1 are average ones for solved instances. The percentage of solved problems is actually relative to the 50% (which is an approximation) of tested instances that are expected to be satisfiable since they are selected at the phase transition. The (average cumulated flips for solved instances)/(percentage of solved problems) ratio is also given in order to conduct a fair comparison when one of the algorithms solves more instances than the other one. MAX-TRIES is 5, MAX-FLIPS is n^2 and $p = 0.5$ for RWS-GSAT. These two last values are the (experimentally) best ones for GSAT with respect to random 3-SAT problems (Parkes & Walser 1996; Selman, Kautz, & Cohen 1993).

Table 1 summarizes the results and shows the good performance of TSAT. Let us stress that the given number of flips corresponds to the cumulated number of performed flips during the successive tries. However, most of the time, TSAT just required one try.

Let us note that the competitiveness of TSAT increases with the number of variables. This is illustrated in Figure 5). Results for 2000 variables are the most significant ones but could not have been inserted in the diagram because of the huge difference (see Table 1).

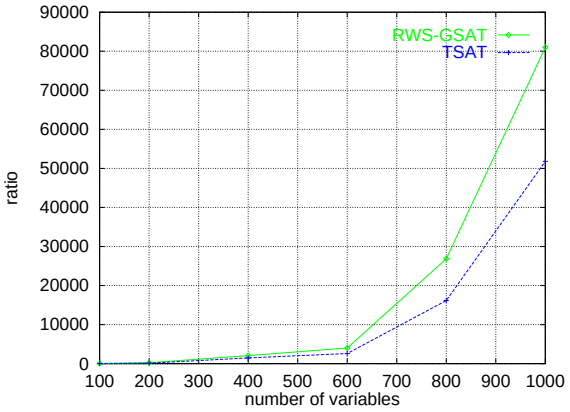


Figure 5: Results for 3-SAT instances at the phase transition obtained by RWS-GSAT and TSAT

Conclusion

TSAT, a basic tabu search algorithm for SAT, has been proposed and compared with Selman et al. Random Walk Strategy GSAT procedure. TSAT makes a systematic use of a tabu list and takes out one of the randomness properties of RWS-GSAT. TSAT proves very competitive in the resolution of many problems, in particular hard random K-SAT instances. Quite surprisingly, the optimal length of the tabu lists for these random problems proves (experimentally) linear with respect to the number of variables. This linearity was quite unexpected, as well as the fact that the length does only depend on the number of variables. This finding is certainly worth further research. Intuitively, the length of the tabu list could be related to some extent to the height of local extrema. We are currently working on that, trying to relate this feature to the nature of really hard

random SAT problems. Also, as TSAT uses a basic form of tabu search, we are currently working on more sophisticated tabu strategies (Glover 1989; 1990) and extending them to other related problems (in this respect, see also the related work by (Battiti & Protasi 1996)). In particular, we are experimenting with TSAT with respect to structured examples (as those suggested in (DIMACS 1993)), using tabu lists whose lengths are dynamically fine-tuned. Another promising path of research consists in extending TSAT into a logically complete technique that keeps the power of local search for satisfiable instances. In this respect, (Mazure, Saïs, & Grégoire 1996) is a first promising step.

Acknowledgments

This work has been supported in part by the Ganymède II project of the Contrat de Plan Etat/Nord–Pas-de-Calais, and by the IUT de Lens. We thank the reviewers for their useful comments.

References

Battiti, R., and Protasi, M. 1996. Reactive Search, a history-based heuristic for MAX-SAT. In *Proceedings of the Workshop on Satifiability Problem*. Siena, Italy.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. 1991. Where the Really Hard Problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, 163–169.

Chvátal, V., and Szemerédi, E. 1988. Many Hard Examples for Resolution. *Journal of the Association for Computing Machinery* 33(4):759–768.

Cook, S. 1971. The Complexity of Theorem Proving Procedures. In *Proceedings of the third Ann. Symp. on Theory of Computing, ACM*, 151–158.

Crawford, J., and Auton, L. 1993. Experimental results on the cross-over point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, 21–27.

Davis, M., and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery* 7:201–215.

DIMACS. 1993. *Proceedings of the Second Challenge, organized by the Center for Discrete Mathematics and Computer Science of Rutgers University.*

Dubois, O., and Carlier, J. 1991. Probabilistic Approach to the Satisfiability Problem. *Journal of Theoretical Computer Science* 81:65–75.

Dubois, O.; André, P.; Boufkhad, Y.; and Carlier, J. 1996. SAT versus UNSAT. In Johnson, D., and Trick, M., eds., *Cliques, Coloring and Satisfiability, Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Science. American Mathematical Society.

Franco, J., and Paull, M. 1983. Probabilistic analysis of the Davis and Putnam Procedure for Solving the Satisfiability Problem. *Journal of Discrete Applied Math.* 5:77–87.

Gent, I., and Walsh, T. 1993. Towards an Understanding of Hill-climbing Procedures for SAT. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, 28–33.

Glover, F. 1989. Tabu search - Part I. *ORSA Journal of Computing* 1(3):190–206.

Glover, F. 1990. Tabu search - Part II. *ORSA Journal of Computing* 2(1):4–32.

Gu, J. 1992. Efficient Local Search for Very Large-Scale Satisfiability Problems. *SIGART Bulletin* 3(1):8–12.

Mazure, B.; Saïs, L.; and Grégoire, E. 1995. A New Local Search Algorithm for SAT: Performance and Analysis. In *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, 127–130.

Mazure, B.; Saïs, L.; and Grégoire, E. 1996. Detecting Logical Inconsistencies. In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, 116–121.

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and Easy Distributions of SAT Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, 459–465.

Parkes, A., and Walser, J. 1996. A Resampling Method to Optimize Maxflips in Local Search: Theory and Application. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, 356–361.

Selman, B.; Kautz, H.; and Cohen, B. 1993. Local Search Strategies for Satisfiability Testing. In *Proceedings of DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, 440–446.

Siegel, P. 1987. Représentation et utilisation des connaissances en calcul propositionnel. Thèse d'État, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II.

Boosting Complete Techniques thanks to Local Search

Article paru dans le journal « *Annals of Mathematics and Artificial Intelligence* », volume 22, pages 319–331, 1998

Co-écrit avec Lakhdar SAÏS et Éric GRÉGOIRE.

Annals of Mathematics and Artifical Inteligence 22 (1998) 319–331 319

Boosting complete techniques thanks to local search methods

Bertrand MAZURE, Lakhdar SAÏS and Éric GRÉGOIRE

CRIL – Université d'Artois, rue de l'Université SP 16, F-62307 LENS Cédex, France
E-mail: {mazure,sais,gregoire}@cril.univ-artois.fr

In this paper, an efficient heuristic allowing one to localize inconsistent kernels in propositional knowledge-bases is described. Then, it is shown that local search techniques can boost the performance of logically complete methods for SAT. More precisely, local search techniques can be used to guide the branching strategy of logically complete Davis and Putnam's like techniques, giving rise to significant performance improvements, in particular when addressing locally inconsistent problems. Moreover, this approach appears very competitive in the context of consistent SAT instances, too.

Keywords: SAT, NP-completeness, local search methods, logical inconsistency, logical completeness

1. Introduction

A fundamental problem in logical knowledge-based representation and automated theorem proving systems lies in the difficulty of checking and handling forms of inconsistency. In particular, any local contradiction in a standard logical knowledge-based system makes it wholly inconsistent: any piece of information (and its contrary) can be inferred from it under sound and complete standard rules of deduction.

At the same time, checking in the general propositional case whether a formula is satisfiable or not -namely, the SAT problem-, can be extremely time-consuming. Indeed, although theoretical and experimental results show good average-case performance for several classes of SAT instances (see e.g. [10]), SAT is NP-complete [3].

Recently, there has been a renewal of interest in understanding the nature of the difficulty of SAT (see e.g. [2] and [9]). At the same time, several authors

have proposed new -but amazingly simple and efficient- algorithms allowing for a breakthrough in the class of computer- solvable consistent SAT instances (see e.g. [17], [18], [7] and [14]). However, since these algorithms are based on local search techniques, they are logically incomplete in that they cannot be used directly to prove in a definitive manner that a formula is inconsistent.

Actually, the most efficient complete techniques (like those derived from Davis and Putnam's one -in short DP- [6]) that are used to prove that a formula is inconsistent exhibit a somewhat limited practical scope with respect to really large and hard unsatisfiable SAT instances. Moreover, we cannot hope for such logically complete techniques to exhibit a polynomial behaviour in all situations unless P = NP.

In this respect, the contribution of this paper is twofold. The behaviour of local search algorithms has been analysed in the context of inconsistent SAT instances. Some recurrent phenomena have been pointed out when locally inconsistent problems were considered, i.e. problems whose inconsistency can be related to one of their subparts. In this context, two main results have been derived.

- First, an efficient heuristic has been discovered, allowing one to localize the inconsistent kernels in many locally inconsistent problems. This result is clearly of prime importance since inconsistent knowledge in real-life applications is often based on contradictions that are just local.
- This heuristic can boost the performance of complete techniques, in particular when dealing with large inconsistent SAT instances. More precisely, by combining this heuristic with the power of local search methods and with the completeness of standard SAT techniques, very good results are obtained with respect to classes of both consistent and inconsistent hard SAT instances.

The paper is organized as follows. First, some technical background about SAT and local search methods is briefly recalled. Then, it is shown that these search methods can help us to localize the inconsistent kernels of large SAT instances. Families of algorithms combining logically complete techniques with local search methods are then proposed. The experimental performance of a basic combination schema is illustrated on hard problems taken from standard benchmarks [7]. The scope of these results is then discussed before further promising paths of research are motivated.

2. Technical Background

SAT consists in checking the satisfiability of a boolean formula in conjunctive normal form (CNF). A CNF formula is a set (interpreted as a conjunction) of clauses, where a clause is a disjunction of literals. A literal is a positive or negated propositional variable.

An interpretation of a boolean formula is an assignment of truth values to its variables. A model is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist.

Recently, there has been a renewal of interest in designing efficient methods for hard SAT instances. On the one hand, several authors have improved logically complete techniques like DP. However, these techniques remain of a somewhat limited practical scope with respect to really large and hard SAT instances. In the sequel, we shall refer to improved versions of DP: namely, a version making use of the FFIS heuristics (i.e. First-Fail In Shortened Clauses) by [16] and C-SAT [8], which appeared to be the most efficient complete technique for SAT at the last DIMACS challenge [7].

On the other hand, logically incomplete techniques based on local search have been shown particularly efficient in proving large and hard consistent problems. Let us now briefly recall one of these methods, namely Selman et al.'s GSAT algorithm [17], [18]. This algorithm performs a greedy local search for a satisfying assignment of a set of propositional clauses. The algorithm starts with a randomly generated truth assignment. It then changes ("flips") the assignment of the variable that leads to the largest increase in the total number of satisfied clauses. Such flips are repeated until either a model is found or a preset maximum number of flips (MAX-FLIPS) is reached. This process is repeated as needed up to a maximum of MAX-TRIES times.

In the sequel, we shall consider two more recent variants of GSAT:

- First, the Random Walk Strategy GSAT [18], which outperforms basic GSAT procedures. This variant of GSAT selects the variable to be flipped in the following way: it either picks with probability p a variable occurring in some unsatisfied clause or follows, with probability $1-p$, the standard GSAT scheme, i.e. makes the best possible local move.
- Second, TSAT [14], which departs from basic GSAT by making an optimized use of a tabu list of variables in order to avoid recurrent flips and thus escape

Algorithm 1. GSAT: basic version

```
Procedure GSAT
Input : a set of clauses S, MAX-FLIPS, and MAX-TRIES
Output : a satisfying truth assignment of S, if found
Begin
   for i := 1 to MAX-TRIES
       I := a randomly generated truth assignment;
       for j := 1 to MAX-FLIPS
           if I satisfies S then return I;
           x := a propositional variable such that a change in its truth
                assignment gives the largest increase (possibly negative)
                in the number of clauses of S that are satisfied by I;
           I := I with the truth assignment of x reversed;
       end for;
   end for;
   return "no satisfying assignment found";
End.
```

from local minima. More precisely, TSAT keeps a fixed length -chronologically-ordered FIFO- list of flipped variables and prevents any of the variables in the list from being flipped again during a given amount of time. TSAT proves very competitive in most situations [14].

These very simple logically incomplete algorithms, which belong to the local search procedures family, are surprisingly efficient in demonstrating that CNF formulas are consistent.

3. Using GSAT-like Techniques to Detect and Locate Local Inconsistencies

In this section, it is shown that GSAT-like techniques can be used to localize inconsistent kernels of SAT instances, although the scope of such logically incomplete algorithms was generally expected to concern consistent problems, only.

The following test has been repeated very extensively and the following results have been obtained extremely often.

When TSAT (or any other GSAT-like algorithm) is run on a SAT instance, the following phenomenon can be observed when the algorithm fails to prove that it is consistent. A trace of TSAT is recorded: for each clause, taking each flip as a step of time, the number of times during which this clause is falsified is updated. A similar trace is recorded for each literal occurring in the SAT instance, counting the number of times it appears in the falsified clauses. Intuitively, it seemed to us that the most often falsified clauses should normally belong to an inconsistent kernel of the SAT instance if this instance is actually inconsistent. Likewise, it seemed to us that the literals that exhibit the highest scores should also take part in this kernel. Actually, these hypotheses prove experimentally correct extremely often.

This phenomenon can be summarized as follows. When GSAT-like algorithms are run on a locally inconsistent SAT instance, then the above counters allow us to split the SAT problem into two parts: a consistent one and an unsatisfiable one. For example, the clausal representation of the well-known inconsistent pigeons-holes problems [19] (8 pigeons; 56 variables and 204 clauses) has been mixed with the 8-queens problems (64 variables and 736 clauses), each problem making use of its own variables. The representation of each problem contains two kinds of clauses: namely, the positive ones (i.e. made of positive literals, only) and the binary negative ones (i.e. made of two negative literals). For example, the basic representation of the pigeons-holes problem asserts, on the one hand, that each pigeon should be in one hole (by means of positive clauses), and, on the other hand, that two pigeons cannot share a hole (using negative clauses). The number of times the different clauses have been falsified is shown in Figure 1. A significant gap can be seen between the scores of the pigeons-holes clauses and the scores of the 8-queens ones. Actually, the two parts of the mixed problem are clearly identified.

Moreover, in this specific case, for each part of the clausal representation (for example, the positive clauses of the pigeons-holes problem), the concerned clauses exhibit extremely close scores. In the diagram, the medium score are mentioned (actually, the four mentioned scores belong to [17189..18003], [639..828], [182..230] and [1..38], respectively for TSAT with MAX-TRIES = 20 and MAX-FLIPS = (number of variables)^2 . The span of these intervals shortens when the computing resources given to TSAT are increased). These close scores show us

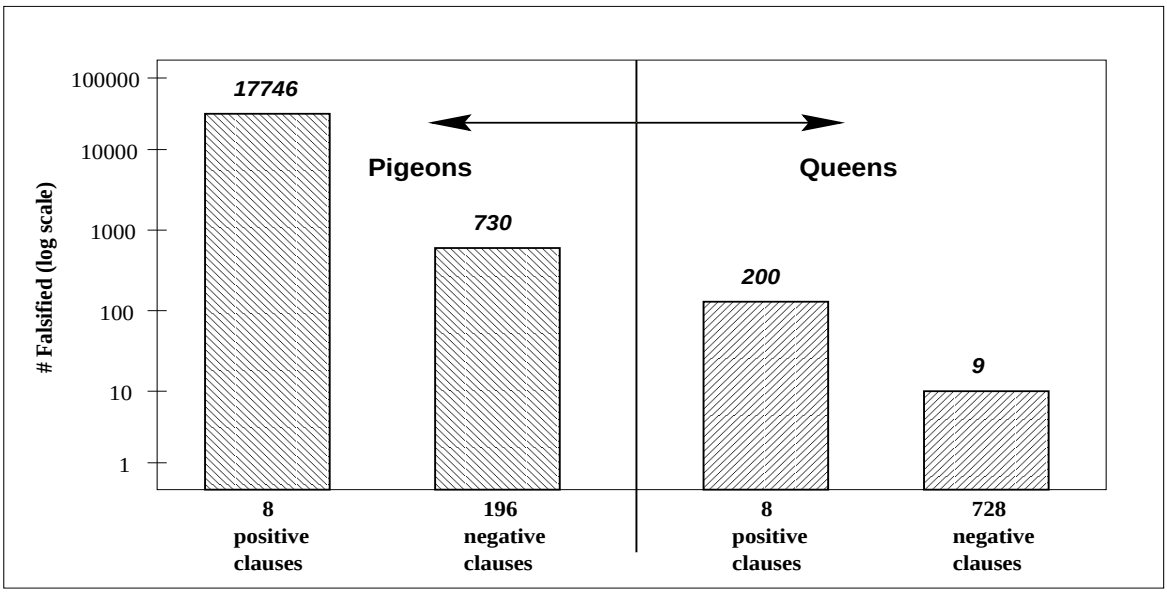


Figure 1. 8-Pigeons-holes + 8-Queens problems

that TSAT also exhibits the complete symmetry of each part of the representation of the pigeons-holes and queens problems. Actually, the trace of GSAT-related algorithms allows us to detect symmetries in SAT instances.

For less symmetrical and non-symmetrical problems, a significant gap between the scores of two parts of the clausal representation is still obtained, differentiating a probable inconsistent kernel from the remaining part of the problem. Let us stress that this phenomena also appear when both the consistent and the inconsistent parts of the SAT instance share the same variables.

Strangely enough, it appears thus that the trace of GSAT-like algorithms delivers the probable inconsistent kernel of locally inconsistent problems and sometimes allows us to detect and locate the presence of symmetries in SAT instances.

4. A Direct Approach: Focus on the Kernel

The most straightforward use of the above discovered feature to solve locally inconsistent problems is the following one. Use GSAT-like algorithms to circumscribe the probable inconsistent kernel. Then, apply complete techniques to this kernel to prove its unsatisfiability and thus, consequently, the inconsistency of the global problem. The scores delivered by the GSAT-like algorithm can be used to guide the branching strategy performed by the complete technique, by selecting

the literals with the highest scores first.

Obviously, this simple schema can only be used when the discovered probable inconsistent kernel is of manageable size and when the gap between the score of the clauses of the kernel and the score of the remaining clauses is large enough. Also, the clauses can be sorted according to their decreasing scores; incremental complete techniques can be applied on them until unsatisfiability is proved. In this respect, clauses outside the discovered kernel could also be taken into account. A somewhat similar approach has been defined independently in [5].

5. A Basic Combination Schema

Let us now describe a basic algorithm using GSAT-like procedures to guide the branching strategy of logically-complete algorithms based on DP.

TSAT [14] is run to deliver the next literal to be assigned by DP. This literal is selected as the one with the highest score as explained above. Such an approach can be seen as using the trace of TSAT as an heuristic for selecting the next literal to be assigned by DP, and a way to extend the partial assignment made by DP towards a model of the SAT instance when this instance is satisfiable. Each time DP needs to select the next variable to be considered, such a call to TSAT can be performed with respect to the remaining part of the SAT instance. This algorithm is given in Algorithm 2.

Algorithm 2. DP + TSAT: basic version

Procedure *DP + TSAT*
Input : a set of clauses S
Output : a satisfying truth assignment of S, if found
or a definitive statement that S is inconsistent
Begin
Unit_propagate(S);
if the empty clause is generated **then return** (false);
else if all variables are assigned **then return** (true)
else begin
if TSAT(S) succeeds **then return** (true)
else begin
p **:=** the most often falsified literal during TSAT;
return (DP+TSAT(S∧p) ∨ DP+TSAT(S∧¬p));

```
                    end;
            end;
  End
```

6. Experimental Results[1]

In this section, experimental results about the application of the above new logically complete algorithm to many classes of problems are presented. Let us stress that our goal in conducting these tests was simply to check the feasibility of using GSAT-like techniques to guide DP. In this respect, a basic form of combination has been tested, which can be improved in many directions as this will be described in the next section.

First, DP+TSAT has been run and compared with DP + FFIS and C-SAT (with default option) on various large inconsistent problems from the DIMACS benchmarks [7]. This benchmark consists of various SAT instances: problems from real-life applications, academic and random ones, many of them being out of reach of the most efficient techniques, in particular complete ones. In Table 1, a significant sample of our extensive experimentations is given, showing the obtained dramatical performance improvement, in particular for classes of inconsistent SAT instances. Also, very good results are obtained for consistent instances; indeed, DP+TSAT is as efficient as local search techniques since DP+TSAT begins with a call to them. Extensive results on DIMACS benchmarks are given in the appendix.

Let us comment a few examples from Table 1.

The BF1355-638 problem (2177 variables and 4768 clauses) is an actual problem from circuit fault analysis proposed by Allen Van Gelder and Yumi Tsuji. C-SAT and DP+FFIS failed to prove that this problem is inconsistent, within 73H and 17H CPU time, respectively. On the other hand, DP+TSAT

[1] All the algorithms mentioned in this paper are implemented in a common platform, available from the authors, written in C under Linux 1.1.53 (except that we reused C-SAT, the original DIMACS'93 implementation by Dubois). We have thus implemented the TSAT, DP+TSAT and DP+FFIS procedures, this latter one being at least as efficient as Rauzy's original one. All experimentations have been conducted on 133 Pentium PCs.

Table 1
DIMACS problems[2]

Instances	Sat	Size		Inc. Ker.[3]		C-Sat	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.	time	assign.	choices	time	assign.	choices	time
AIM series:												
1_6-no-3	No	100	160	51	57	13s32	3E+07	2E+06	214s71	178	16	0s26
1_6-yes1-2	Yes	100	160	***	***	0s00	495858	30052	4s39	77	6	0s08
2_0-no-1	No	100	200	18	19	313s24	4E+07	2E+06	349s52	46	5	0s10
2_0-yes1-1	Yes	100	200	***	***	8s54	706388	31274	7s41	81	8	0s15
1_6-no-1	No	200	320	52	55	20116s30	***	***	>8h	240	16	0s58
1_6-yes1-3	Yes	200	320	***	***	>7h	***	***	>9h	232	11	0s32
2_0-no-3	No	200	400	35	37	>20h	***	***	>15h	120	10	0s42
2_0-yes1-1	Yes	200	400	***	***	>8h	2E+09	7E+07	21859s45	291	27	1s21
1_6-no-1	No	50	80	20	22	0s19	12072	895	0s09	72	8	0s06
1_6-yes1-1	Yes	50	80	***	***	0s07	1540	84	0s01	37	6	0s05
2_0-no-1	No	50	100	21	22	0s30	54014	2759	0s43	52	5	0s05
2_0-yes1-1	Yes	50	100	***	***	0s19	2878	176	0s03	11	3	0s03
BF series:												
0432-007	No	1040	3668	674	1252	463s67	9E+08	6E+06	19553s44	115766	870	85s25
1355-075	No	2180	6778	82	185	88035s05	317628	2047	18s88	4602	28	26s23
1355-638	No	2177	4768	83	154	>73h	***	***	>17h	6192	32	32s57
2670-001	No	1393	3434	79	139	11s45	***	***	>25h	490692	4822	519s40
SSA series:												
0432-003	No	435	1027	306	320	1s70	133794	1570	1s79	1338	16	0s80
2670-130	No	1359	3321	552	669	2053s72	***	***	>33h	2E+07	79426	8040s64
2670-141	No	986	2315	579	1247	3689s65	3E+08	2E+06	6350s77	1E+07	92421	6639s44
7552-038	Yes	1501	3575	***	***	195s75	***	***	>13h	29	1	0s34
7552-158	Yes	1363	3034	***	***	97s59	1639	78	0s19	12	1	0s29
7552-159	Yes	1363	3032	***	***	98s82	1557	84	0s21	12	1	0s25
7552-160	Yes	1391	3126	***	***	159s75	1457	76	0s18	1	1	0s30

takes 32 sec. only.

The AIM200-1_6-yes1-3 problem (200 variables and 320 clauses) by [13] is a 3-SAT instance. All AIM problems exhibit exactly one model when satisfiable. DP+TSAT proves within 0.32 sec. that the above mentioned AIM problem is satisfiable while DP + FFIS and C-SAT gave up after 9 and 7H, respectively.

The AIM200-2_0-no-3 problem (200 variables and 400 clauses) by [13] is an inconsistent 3-SAT instance. DP+TSAT takes 0.42 sec. to prove that this problem is inconsistent, whereas we gave up with DP+FFIS and C-SAT after 15 and 20 H, respectively.

[2] In the table, "$> n$ H" means that we gave up after the problem had not been solved within n hours of CPU time.

[3] "Inc. Ker." in the table means "Inconsistent Kernel".

7. Natural Optimizations

Although the above mentioned results are extremely positive, it should be clear that the tests were only conducted for checking the feasibility of mixing local search techniques with logically complete methods. We did not try to optimize the way this mixing is performed. Further significant performance improvements can be expected by fine-tuning the parameters of the involved local search techniques and by defining an optimal balance between the time spent by DP and by the local search techniques. More precisely, the following natural optimizations can be envisioned.

- In DP+TSAT, the literal that exhibits the highest score in the trace of TSAT was selected to guide DP. In this respect, a call to TSAT is performed each time DP needs to select a literal to assign. At the opposite, just one call to TSAT can be made and decreasing scores of literals can be used to guide DP at each step. Between these two extreme points of view, an optimal attitude must be found, limiting the number of calls to GSAT-like algorithms. This optimal balance should depend on both the form of the trace given by this GSAT-like algorithm and on the point reached in the search tree by DP.
- A second possible optimization lies in the fine-tuning of the local search parameters with respect to the remaining problem left by DP.

8. Actual Scope of These Techniques

Clearly, the results presented in this paper concern consistent and locally inconsistent problems, i.e. problems whose inconsistency can be related to one of their subparts. Let us define the dual concepts of local and global inconsistency for SAT instances.

Definition 1.

A SAT instance S is *globally inconsistent*

iff

S is inconsistent and $\forall S' \subset S : S'$ is consistent.

When an inconsistent SAT instance S is not globally inconsistent, it is said to be locally inconsistent.

In this paper, locally inconsistent SAT instances have been considered. Indeed, such a form of inconsistency is of prime importance in actual applications. Very often, inconsistency is due to the accidental presence of a few pieces of contradictory information about a given subject. Clearly, several measures of locality for inconsistency can be defined, making use of e.g. the (size of the inconsistent kernel)/(size of the SAT instance) ratio. We are currently analyzing how the form and properties of the trace of local search techniques can be experimentally related to graded notions of locality for inconsistency. Let us stress that the techniques presented in this paper do not require this ratio to be negligible, as the ratio of the combination of the pigeons-holes and 8-queens problems illustrates it. Also, the size of the kernel need not be small.

In Table 1 and in the Appendix, we give the size of (one of) the globally inconsistent kernels for each inconsistent problem that DP+TSAT managed to solve. To obtain this information, first we used the trace of TSAT to get a first inconsistent kernel using a dichotomy-like approach. Then, using complete methods, we reduced the kernel to a subpart that we proved to be globally inconsistent. Let us stress that most often this resulting kernel proved to be close to the initial one, showing once again the relevance of the heuristic described in this paper.

As expected, DP+TSAT managed to prove inconsistent problems, where one globally inconsistent kernel is of moderate size. Let us note that sometimes DP+FFIS and C-SAT did not manage to solve them: see for example the AIM-200-2_0-no-3 where the size of one globally inconsistent kernel is just made of 37 clauses refering to 35 variables. Interestingly enough, DP+TSAT also gave rise to performance improvement with respect to problems involving a large globally inconsistent kernel: see for example the bf0432-007 problem that DP+TSAT proved to be inconsistent, addressing a globally inconsistent kernel made of 1252 clauses making use of 674 variables.

Obviously, large globally inconsistent problems are the most difficult to handle; they are often generated in an artificial manner since they are scarce in real life applications (see e.g. the pigeons-holes problem [4], Tseitin and Urqhart's formulas [19], etc.). Really significant progress in dealing with large globally inconsistent problems requires a better understanding of their nature and properties. However, the results presented in this paper also apply to globally inconsistent problems; at least to some extent, as we have illustrated it earlier when we discussed the size of the discovered inconsistent kernels of DIMACS benchmarks. Also, when DP+TSAT (which is once again a rough non-optimized combination

schema) did not give rise to better CPU-time on globally inconsistent problems like Dubois' ones, it allowed smaller search trees to be generated. In the same vein, we hope that some progress could be made with respect to hard random problems. In particular, we hope that some little progress could be obtained in the treatment of inconsistent K-SAT instances at the transition phase in the fixed-length clause model [9], at least, as far as locally inconsistent instances are still actually under consideration. On the other hand, we are very optimistic in making good progress in solving inconsistent random K-SAT instances at the right of the transition phase.

9. Conclusion

The contribution of this paper is twofold. On the one hand, an efficient heuristic-based technique has been proposed, allowing one to detect and locate local inconsistencies in sets of propositional clauses. We think that such a technique should be useful with respect to many computer science domains. For example, this should make the handling of local inconsistencies in logical knowledge bases possible. On the other hand, using local search techniques, the feasibility of boosting complete techniques to prove SAT instances has been demonstrated, in particular large locally inconsistent ones that were out of reach of previous approaches. Moreover, the technique presented in this paper also appears competitive for solving classes of consistent SAT instances. Additionally, further possible optimizations that could lead to additional significant performance improvements have been discussed.

Acknowledgements

This work has been supported by the Ganymède II project of the Contat de Plan Etat/Nord–Pas-de-Calais, by the MESR (Ministère de l'Enseignement Supérieur et de la Recherche) and by the IUT de Lens. We express our gratitute to our colleague J.-L. Coquidé for his help in providing us with access to computing facilities at the IUT de Lens, allowing our extensive tests to be performed.

References

[1] P. Cheeseman and B. Kanefsky and W.M. Taylor, Where the Really Hard Problems are, in: *Proc. IJCAI-91*, pp. 163-169, 1991.

[2] V. Chvátal and E. Szemerédi, Many Hard Examples for Resolution, in: *Journ. of the ACM*, vol. 33, no. 4, pp. 759-768, 1988.

[3] S. Cook, The Complexity of Theorem-Proving Procedures, in: *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pp. 151-158, 1971.

[4] S. Cook, A Short Proof of the Pigeon Hole Principle Using Extended Resolution, in: *SIGACT News*, vol. 8, pp. 28-32, 1976.

[5] J.M. Crawford, Solving Satisfiability Problems Using a Combination of Systematic and Local Search, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[6] M. Davis, H. Putnam, A Computing Procedure for Quantification Theory, in: *Journ. of the ACM*, vol. 7, pp. 201-215, 1960.

[7] DIMACS, Second challenge organized by the Center for Discrete Mathematics and Computer Science of Rutgers University in 1993 (The benchmarks used in our tests can be obtained by anonymous ftp from Rutgers University DIMACS Center: ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf).

[8] O. Dubois and P. André and and Y. Boufkhad and J. Carlier, SAT versus UNSAT, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[9] O. Dubois and J. Carlier, Probabilistic Approach to the Satisfiability Problem, in: *Theoretical Computer Science*, vol. 81, pp. 65-75, 1991.

[10] J. Franco and M. Paull, Probabilistic Analysis of the Davis and Putnam Procedure for Solving the Satisfiability Problem, in: *Discrete Applied Math.*, vol. 5, pp. 77-87, 1983.

[11] I.P. Gent and T. Walsh, Towards an Understanding of Hill-climbing Procedures for SAT, in: *Proc. AAAI-93*, pp.28-33, 1993.

[12] I.P. Gent and T. Walsh, The SAT Phase Transition, in: *Proc. ECAI-94*, pp. 105-109, 1994.

[13] K. Iwama and E. Miyano, Test-Case Generation with Proved Securities, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[14] B. Mazure and L. Saïs and É. Grégoire, Tabu Search for SAT, in: *Proc. AAAI-97*, pp. 281-285, July 1997. (A preliminary version appeared as: TWSAT: a New Local Search Algorithm for SAT. Performance and Analysis, in: *CP95 Workshop on Studying and Solving Really Hard Problems*, Cassis (France), September 1995.)

[15] D. Mitchell and B. Selman and H. Levesque, Hard and Easy Distributions of SAT Problems, in: *Proc. AAAI-92*, pp. 459-465, 1992.

[16] A. Rauzy, On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT, in: *LaBRI Technical Report 806-94*, Université de Bordeaux 1, 1994.

[17] B. Selman and H. Levesque and D. Mitchell, A New Method for Solving Hard Satisfiability Problems, in: *Proc. AAAI-92*, pp. 440-446, 1992.

[18] B. Selman and H.A. Kautz and B. Cohen, Local Search Strategies for Satisfiability Testing, in: *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[19] G.S. Tseitin, On the Complexity of Derivations in Propositional Calculus, in: Slisenko A.O. (ed.), *Structures in Constructive Mathematics, Part II*, pp. 115-125, 1968.

Appendix

In the next table:

- "Inc. Ker." means "Inconsistent Kernel".
- "$> n$ H" means that we gave up after the problem had not been solved within n hours of CPU time,
- "***" in:
  - "Inc. Ker." column means that the instance is satisfiable,
  - other columns means that the method fails to solve the instance,
- "???" in Inc. Ker. column means that DP+TSAT fails to detect an inconsistent kernel.

Table 2
DIMACS problems

Instances	Sat	Size		Inc. Ker.		C-Sat	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.	time	assign.	choices	time	assign.	choices	time
Dubois series:												
dubois10	No	30	80	30	80	0s22	20564	2063	0s15	8728	641	1s30
dubois11	No	33	88	33	88	0s44	41044	4111	0s31	21800	1691	3s34
dubois12	No	36	96	36	96	0s87	82004	8207	0s59	25228	1835	4s48
dubois13	No	39	104	39	104	1s79	163924	16399	1s18	47236	3091	8s69
dubois14	No	42	112	42	112	3s67	327764	32783	2s49	99888	7693	14s39
dubois15	No	45	120	45	120	7s39	655444	65551	5s15	147180	9507	26s45
dubois16	No	48	128	48	128	15s22	1E+06	131087	10s09	303108	20733	65s97
dubois17	No	51	136	51	136	31s70	3E+06	262159	19s69	348068	23167	76s86
SSA series:												
0432-003	No	435	1027	306	320	1s70	133794	1570	1s79	1338	16	0s80
2670-130	No	1359	3321	552	669	2053s72	***	***	>33h	2E+07	79426	8040s64
2670-141	No	986	2315	579	1247	3689s65	3E+08	2E+06	6350s77	1E+07	92421	6639s44
6288-047	No	10410	34238	???	???	>24h	***	***	>24h	***	***	>24h
7552-038	Yes	1501	3575	***	***	195s75	***	***	>13h	29	1	0s34
7552-158	Yes	1363	3034	***	***	97s59	1639	78	0s19	12	1	0s29
7552-159	Yes	1363	3032	***	***	98s82	1557	84	0s21	12	1	0s25
7552-160	Yes	1391	3126	***	***	159s75	1457	76	0s18	1	1	0s30
BF series:												
0432-007	No	1040	3668	674	1252	463s67	9E+08	6E+06	19553s44	115766	870	85s25
1355-075	No	2180	6778	82	185	88035s05	317628	2047	18s88	4602	28	26s23

Instances	Sat	Size		Inc.	Ker.	C-Sat	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.	time	assign.	choices	time	assign.	choices	time
1355-638	No	2177	4768	83	154	>73h	***	***	>17h	6192	32	32s57
2670-001	No	1393	3434	79	139	11s45	***	***	>25h	490692	48220	519s4
AIM series (100 variables):												
1_6-no-1	No	100	160	43	47	0s89	7E+06	323296	50s30	174	13	0s22
1_6-no-2	No	100	160	46	52	0s09	3E+06	167445	23s38	178	19	0s34
1_6-no-3	No	100	160	51	57	13s32	3E+07	2E+06	214s71	178	16	0s26
1_6-no-4	No	100	160	43	48	0s05	1E+07	728908	97s59	126	14	0s25
1_6-yes1-1	Yes	100	160	***	***	0s04	135966	6663	1s09	138	8	0s12
1_6-yes1-2	Yes	100	160	***	***	0s00	495858	30052	4s39	77	6	0s08
1_6-yes1-3	Yes	100	160	***	***	0s04	604	34	0s01	112	12	0s18
1_6-yes1-4	Yes	100	160	***	***	0s04	159646	7707	1s46	124	6	0s08
2_0-no-1	No	100	200	18	19	313s24	4E+07	2E+06	349s52	46	5	0s10
2_0-no-2	No	100	200	35	39	72s19	3E+07	1E+06	294s09	108	9	0s20
2_0-no-3	No	100	200	25	27	274s14	9E+06	394649	80s77	68	6	0s12
2_0-no-4	No	100	200	26	31	0s05	3E+07	1E+06	233s29	80	9	0s19
2_0-yes1-1	Yes	100	200	***	***	8s54	706388	31274	7s41	81	8	0s15
2_0-yes1-2	Yes	100	200	***	***	1s85	238794	10305	2s79	91	10	0s20
2_0-yes1-3	Yes	100	200	***	***	5s19	138	10	0s00	84	6	0s12
2_0-yes1-4	Yes	100	200	***	***	0s70	270	18	0s00	163	9	0s13
3_4-yes1-1	Yes	100	340	***	***	0s10	996	30	0s02	1	2	0s08
3_4-yes1-2	Yes	100	340	***	***	0s15	2366	74	0s06	0	1	0s00
3_4-yes1-3	Yes	100	340	***	***	0s10	5484	193	0s14	0	1	0s01
3_4-yes1-4	Yes	100	340	***	***	0s10	196	17	0s01	0	1	0s02
6_0-yes1-1	Yes	100	600	***	***	0s15	100	4	0s01	0	1	0s01
6_0-yes1-2	Yes	100	600	***	***	0s27	248	7	0s01	0	1	0s00
6_0-yes1-3	Yes	100	600	***	***	0s12	224	11	0s01	0	1	0s01
6_0-yes1-4	Yes	100	600	***	***	0s12	212	11	0s01	0	1	0s00
AIM series (200 variables):												
1_6-no-1	No	200	320	52	55	20116s30	***	***	>8h	240	16	0s58
1_6-no-2	No	200	320	77	80	0s17	***	***	>15h	262	24	0s88
1_6-no-3	No	200	320	77	83	0s70	***	***	>8h	302	33	1s15
1_6-no-4	No	200	320	44	46	0s04	***	***	>17h	184	16	0s61
1_6-yes1-1	Yes	200	320	***	***	>12h	210	10	0s01	222	10	0s25
1_6-yes1-2	Yes	200	320	***	***	0s04	682	34	0s02	240	14	0s44
1_6-yes1-3	Yes	200	320	***	***	>7h	***	***	>9h	232	11	0s32
1_6-yes1-4	Yes	200	320	***	***	0s04	6E+08	3E+07	5567s85	244	13	0s42
2_0-no-1	No	200	400	49	53	>7h	***	***	>15h	136	11	0s47
2_0-no-2	No	200	400	46	50	>20h	***	***	>8h	186	15	0s67
2_0-no-3	No	200	400	35	37	>20h	***	***	>15h	120	10	0s42
2_0-no-4	No	200	400	36	42	0s07	***	***	>8h	144	13	0s55
2_0-yes1-1	Yes	200	400	***	***	>8h	2E+09	7E+07	21859s45	291	27	1s21
2_0-yes1-2	Yes	200	400	***	***	8273s44	2E+08	7E+06	2809s87	290	29	1s18
2_0-yes1-3	Yes	200	400	***	***	1197s97	4960	217	0s09	444	20	0s98

Instances	Sat	Size		Inc.	Ker.	C-Sat	DP+FFIS			DP+TSAT		
		Var.	Cla.	Var.	Cla.	time	assign.	choices	time	assign.	choices	time
2_0-yes1-4	Yes	200	400	***	***	54296s32	272472	11783	4s79	319	27	1s07
3_4-yes1-1	Yes	200	680	***	***	0s52	210956	5409	6s88	0	1	0s06
3_4-yes1-2	Yes	200	680	***	***	0s29	8896	272	0s37	0	1	0s07
3_4-yes1-3	Yes	200	680	***	***	0s70	1302	36	0s05	2	3	0s22
3_4-yes1-4	Yes	200	680	***	***	0s35	267786	6270	8s44	0	1	0s05
6_0-yes1-1	Yes	200	1200	***	***	0s57	6110	75	0s25	0	1	0s01
6_0-yes1-2	Yes	200	1200	***	***	0s22	10284	134	0s47	0	1	0s01
6_0-yes1-3	Yes	200	1200	***	***	0s54	16704	214	0s72	0	1	0s01
6_0-yes1-4	Yes	200	1200	***	***	0s34	16412	232	0s72	0	1	0s04
AIM series (50 variables):												
1_6-no-1	No	50	80	20	22	0s19	12072	895	0s09	72	8	0s06
1_6-no-2	No	50	80	28	32	0s02	11408	782	0s10	86	9	0s07
1_6-no-3	No	50	80	28	31	0s07	54610	4525	0s41	92	12	0s09
1_6-no-4	No	50	80	18	20	0s02	7230	447	0s05	56	7	0s06
1_6-yes1-1	Yes	50	80	***	***	0s07	1540	84	0s01	37	6	0s05
1_6-yes1-2	Yes	50	80	***	***	0s02	5674	384	0s05	50	3	0s02
1_6-yes1-3	Yes	50	80	***	***	0s04	50	5	0s01	50	4	0s04
1_6-yes1-4	Yes	50	80	***	***	0s02	70	5	0s01	1	2	0s01
2_0-no-1	No	50	100	21	22	0s30	54014	2759	0s43	52	5	0s05
2_0-no-2	No	50	100	28	30	0s07	19342	974	0s17	80	7	0s07
2_0-no-3	No	50	100	22	28	0s22	15254	814	0s13	76	6	0s06
2_0-no-4	No	50	100	18	21	0s02	30034	1645	0s24	60	6	0s06
2_0-yes1-1	Yes	50	100	***	***	0s19	2878	176	0s03	11	3	0s03
2_0-yes1-2	Yes	50	100	***	***	0s07	432	29	0s01	0	1	0s00
2_0-yes1-3	Yes	50	100	***	***	0s10	7616	446	0s07	12	3	0s02
2_0-yes1-4	Yes	50	100	***	***	0s04	92	8	0s00	0	1	0s00
3_4-yes1-1	Yes	50	170	***	***	0s04	496	20	0s01	0	1	0s00
3_4-yes1-2	Yes	50	170	***	***	0s07	406	13	0s01	0	1	0s00
3_4-yes1-3	Yes	50	170	***	***	0s02	378	16	0s01	0	1	0s01
3_4-yes1-4	Yes	50	170	***	***	0s05	282	13	0s00	0	1	0s00
6_0-yes1-1	Yes	50	300	***	***	0s09	124	4	0s01	0	1	0s00
6_0-yes1-2	Yes	50	300	***	***	0s07	282	8	0s01	0	1	0s01
6_0-yes1-3	Yes	50	300	***	***	0s09	82	4	0s01	0	1	0s00
6_0-yes1-4	Yes	50	300	***	***	0s07	76	4	0s01	0	1	0s00

Integrating Conflict Driven Clause Learning to Local Search

Article paru dans les actes électroniques de « *International Workshop on Local Search Techniques in Constraint Satisfaction* », affilié à la conférence CP, Lisbonne, Portugal, septembre 2009.

Co-écrit avec Gilles AUDEMARD, Jean-Marie LAGNIEZ et Lakhdar SAÏS.

Integrating Conflict Driven Clause Learning to Local Search

Gilles Audemard Jean-Marie Lagniez Bertrand Mazure Lakhdar Saïs *

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{audemard,lagniez,mazure,sais}@cril.fr

This article introduces SATHYS (*SAT HYbrid Solver*), a novel hybrid approach for propositional satisfiability. It combines local search and conflict driven clause learning (CDCL) scheme. Each time the local search part reaches a local minimum, the CDCL is launched. For SAT problems it behaves like a tabu list, whereas for UNSAT ones, the CDCL part tries to focus on minimum unsatisfiable sub-formula (MUS). Experimental results show good performances on many classes of SAT instances from the last SAT competitions.

1 Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is a central issue in many computer science and artificial intelligence domains, like theorem proving, planning, non-monotonic reasoning, VLSI correctness checking. These last two decades, many approaches have been proposed to solve large SAT instances, based on logically complete or incomplete algorithms. Both local-search techniques [29, 28, 18] and elaborate variants of the Davis-Putnam-Loveland-Logemann DPLL procedure [6] [27, 8], called modern SAT solvers, can now solve many families of hard SAT instances. These two kinds of approaches present complementary features and performances. Modern SAT solvers are particularly efficient on the industrial SAT category while local search performs better on random SAT instances.

Consequently, combining stochastic local search (SLS) and conflict driven clause learning (CDCL) solvers seems promising. Note that it was pointed as a challenge by Selman *et al.* [30] in 1997. Such methods should exploit the quality and differences of both approaches. Furthermore, the perfect hybrid method has to outperform both local search and CDCL solvers. A lot of attempts have been done last decade [4]. These different attempts will be discussed in section 3.

In this paper, we propose another hybridization of local search and modern SAT solver, named SATHYS (*SAT HYbrid Solver*). The local search solver is the main one. Each time it reaches a local minimum, the CDCL part is called and assigns some variables. This part of our solver is expected to have different behaviours depending on the kind of formula to solve. In case of a satisfiable one, the CDCL part can be seen as a tabu list [12] in order to protect good variables and avoid to reach the same minimum quickly. On the other hand, for unsatisfiable formulas, it tries to focus the search on minimum unsatisfiable sub-formulas (MUS) [9, 14, 15], allowing to concentrate on a small part of the whole formula. Like this, the CDCL component of SATHYS is used as a strategy to escape from local minimum.

The rest of the paper is organized as follows. Section 2 introduces different notions necessary for understanding the rest of the paper. Section 3 discusses different hybrid methods. Section 4 gives the

*supported by ANR UNLOC project ANR08-BLAN-0289-01

LSCS 2009
EPTCS , 2009, pp. 1–14, doi:http://dx.doi.org/10.4204/EPTCS..10.4204/EPTCS..

insights of our method. In section 5, we give the details and algorithms of SATHYS. Before a conclusion, section 6 provides different experiments.

2 Preliminary definitions and technical background

2.1 Definitions

Let us give some necessary definitions and notations. Let $V = \{x_1 ... x_n\}$ be a set of boolean variables, a literal ℓ is a variable x_i or its negation $\overline{x_i}$. A clause is a disjunction of literals $c_i = (\ell_1 \vee \ell_2 ... \vee \ell_{n_i})$. A unit clause is a clause with only one literal. A formula Σ is in conjunctive normal form (CNF) if it is a conjunction of clauses $\Sigma = (c_1 \wedge c_2 ... \wedge c_m)$. The set of literals appearing in Σ is denoted $\mathscr{V}_\Sigma$. An interpretation $\mathscr{I}$ of a formula Σ associates a value $\mathscr{I}(x)$ to variables in the formula. An interpretation is *complete* if it gives a value to each variable $x \in \mathscr{V}_\Sigma$, otherwise it is said *partial*. A clause, a CNF formula and an interpretation can be conveniently represented as sets. A *model* of a formula Σ, denoted $\mathscr{I} \models \Sigma$, is an interpretation $\mathscr{I}$ which satisfies the formula Σ i.e. satisfies each clause of Σ. Then, we can define the SAT decision problem as follows: is there an assignment of values to the variables so that the CNF formula Σ is satisfied?
Let us introduce some additional notations.

- The negation of a formula Γ is denoted $\overline{\Gamma}$
- $\Sigma_{|_\ell}$ denotes the formula Σ simplified by the assignment of the literal ℓ to true. This notation is extended to interpretations: Let $\mathscr{P} = \{\ell_1, ..., \ell_n\}$ be an interpretation, $\Sigma_{|_\mathscr{P}} = (...(\Sigma|_{\ell_1})...|_{\ell_n})$;
- Σ^* denotes the formula Σ simplified by unit propagation;
- $\models_*$ denotes logic deduction by unit propagation: $\Sigma \models_* l$ means that the literal x is deducted by unit propagation from Σ i.e. $(\Sigma \wedge \overline{\ell})^* = \bot$. One notes $\Sigma \models_* \bot$ if the formula is unsatisfiable by unit propagation.
- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the opposite literal $\neg x$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$. A resolvent is called *tautological* when it contains opposite literals.

2.2 Local Search Algorithms

Local search algorithms for SAT problems use a stochastic walk over complete interpretations of Σ. At each *step* (or *flip*), they try to reduce the number of unsatisfiable clauses (usually called a descent). The next complete interpretation is chosen among the neighbours of the current one (they differ only on one literal value). A local minimum is reached when no descent is possible. One of the key point of stochastic local search algorithms is the method used to escape from local minimum. For lack of space, we can not provide a general algorithm of local search solver. For more details, the reader will refer to [19].

2.3 CDCL solvers

Algorithm 1 shows the general scheme of a CDCL solver (due to lack of space, we can not provide details for all subroutines). A typical branch of a CDCL solver is a sequence of decisions, followed by propagations, repeated until a conflict is reached. Each decision literal (lines 18–20) is assigned at a given decision level (dl), deducted literals (by unit propagation) have the same decision level. If all variables

Algorithm 1: CDCL solver

```
   Input: a CNF formula Σ
   Output: SAT or UNSAT
 1 I = ∅ ;                                          /* interpretation */
 2 dl = 0 ;                                         /* decision level */
 3 x_c = 0 ;                                  /* number of conflicts */
 4 while (true) do
 5   γ = BCP(Σ,I);
 6   if (γ!=null) then
 7     x_c = x_c + 1;
 8     β =conflictAnalysis(Σ,I,c);
 9     bl = computeBackjumpingLevel(γ,I);
10     if (bl < 0) then return UNSAT;
11     Σ = Σ ∪ {γ};
12     if (restart()) then bl = 0;
13     backjump(Σ,I,bl);
14     dl = bl;
15   else
16     if (all variables are instanciated) then
17       return SAT;
18     ℓ = chooseDecisionLiteral(Σ);
19     dl = dl + 1;
20     I = I ∪ {ℓ};
21
22 end
```

are assigned, then $\mathscr{I}$ is a model of Σ (lines 16–17). Each time a conflict is reached by unit propagation (then γ is the conflict clause) A nogood β is computed (line 8) using a given scheme, usually the first-UIP (Unique Implication Point) one [31] and a backjump level is computed. At this point, It may have proved the unsatisfiability of the formula Σ. If it is not the case, the nogood β is added to the clause database and backjump is done (lines 11–14). Finally, sometimes CDCL solvers enforce restarts (different strategies are possible [20]). In this case, one backjump in the top of the search tree.

2.4 Muses

Minimum unsatisfiable sub-formulas (MUS) of a CNF formula represent the smallest explanations for the inconsistency in term of the number of clauses. MUS are very important in order to circumscribe and highlight the source of contradiction of a given formula. Formally, one has:

Definition 1 *Let Σ be a CNF formula. A MUS Γ of Σ is a set of clauses such that:*

1. $\Gamma \subseteq \Sigma$;
2. Γ *is unsatisfiable;*
3. $\forall \Delta \subset \Gamma, \Delta$ *is satisfiable.*

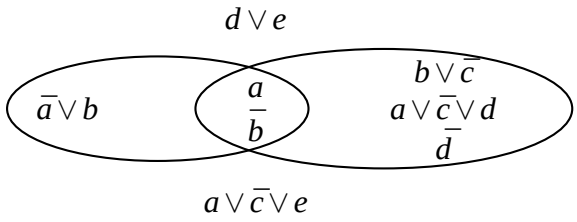


Figure 1: All MUS of the formula Σ (example 1)

Example 1 *Let* $\Sigma = \{\bar{d} \vee e,\ b \vee \bar{c},\ \bar{d},\ \bar{a} \vee b,\ a,\ a \vee \bar{c} \vee e,\ \bar{a} \vee c \vee d,\ \bar{b}\}$ *be a CNF formula. Figure 1 represents all MUS of* Σ*.*

Due to unsatisfiability of MUS, one has the following property:

Proposition 1 *Let* Σ *be an unsatisfiable CNF formula,* Γ *a MUS of* Σ*.*

$$\forall \mathscr{I} \text{ an interpretation over } \mathscr{V}_\Sigma,\ \exists c \in \Gamma \text{ such that } \mathscr{I} \not\models c$$

Let us consider a CNF formula Σ and a complete interpretation $\mathscr{I}_c$. We say that the literal ℓ satisfies (resp. falsifies) a clause $\beta \in \Sigma$ if $\ell \in \mathscr{I}_c \cap \beta$ (resp. $\ell \in \mathscr{I}_c \cap \bar{\beta}$). We note $\mathscr{L}^+_{\mathscr{I}_c}(\beta)$ (resp. $\mathscr{L}^-_{\mathscr{I}_c}(\beta)$), the set of literals satisfying (resp. falsifying) a clause β. The following definitions were introduced in [13].

Definition 2 (once-satisfied clause) *A clause* β *is said* once-satisfied *by an interpretation* $\mathscr{I}_c$ *on literal* z *if* $\mathscr{L}^+_{\mathscr{I}_c}(\beta) = \{z\}$.

Definition 3 (critical and linked clauses) *Let* $\mathscr{I}_c$ *be a complete interpretation. A clause* α *is critical wrt* $\mathscr{I}_c$ *if* $|\mathscr{L}^+_{\mathscr{I}_c}(\alpha)| = 0$ *and* $\forall \ell \in \alpha$, $\exists \alpha' \in \Sigma$ *with* $\bar{\ell} \in \alpha'$ *and* $|\mathscr{L}^+_{\mathscr{I}_c}(\alpha')| = 1$*. Clauses* α' *are linked to* α *for the interpretation* $\mathscr{I}_c$*.*

Example 2 *Let* $\Sigma = (\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (a \vee \bar{b}) \wedge (b \vee \bar{c}) \wedge (c \vee \bar{a})$ *be a formula and* $\mathscr{I}_c = \{a, b, c\}$ *an interpretation. The clause* $\alpha_1 = (\bar{a} \vee \bar{b} \vee \bar{c})$ *is critical. The other clauses of* Σ *are linked to* α_1 *for* $\mathscr{I}_c$*.*

The following properties was proposed and exploited in order to compute MUS by [13].

Proposition 2 *In a minimum (local or global), the set of falsified clauses are critical.*

Proposition 3 *In a minimum (local or global), at least one of clause of each MUS is critical.*

3 Related Works

As it was suggested in the introduction, a lot of different approaches have been proposed to combine local search and DPLL based ones. One can divide such hybridizations in three different categories depending on the kind of the main solver. First, the main solver can be the SLS one. In that case, DP is used in order to help SLS [26, 5, 1, 17]. All of these approaches use the local search component as an assistance for the heuristic choice for variable assignment. Some of them try to focus the search on the unsatisfiable part of the formula [26], others on the satisfiable one [2, 1]. Furthermore, this step can be achieved before the search [5] or dynamically at each decision nodes [26].

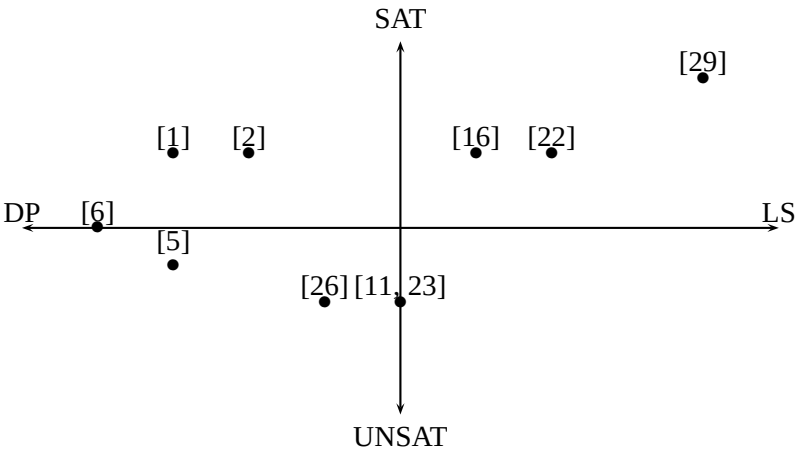


Figure 2: Classification

The second category of hybridizations is the opposite, that is, the SLS solver is the core of the method and the DPLL one helps it [16, 22]. In [16], the DPLL solver is used in order to find dependencies between variables. Then, the local search framework is called on a subset of variables (the independent ones). Whereas, in [22] (note that this method is for constraint satisfaction problems), the local search engine is used to find a promising partial interpretation.

Finally, the last category contains hybrid solvers where the both engines work together [11, 23]. The second method is an improvement of the first one. The local search tries to find a solution. After some time, it stops and sends all falsified clauses by the current interpretation to the CDCL part. This last one has the responsibility to find a model to this sub-formula. If it proves unsatisfiable, then the whole instance is unsatisfiable too.

We propose in Figure 2 a classification of all of these approaches. The X-axis corresponds to the kind of search. For example, DPLL is at the left, whereas walksat is at the right. The Y-axis corresponds to the ability to solve SAT and/or UNSAT formulas. Then, walksat is at the top of the classification. Methods introduced above are located in this graph. Of course, this classification is subjective and and it can be subject of discussion. It is here to help the reader to understand all of these approaches.

4 Intuition

In this section, we provide insights of our hybrid approach SATHYS. They are related to the satisfiability or unsatisfiability of the formulas. First note that the SLS engine is the core of our method. Then, the Local search part tries to find a solution. When a local minimum is reached, the CDCL part of the solver is launched. It works like a tabu list in case of satisfiable formula and tries to focus on MUS for unsatisfiable ones. Let us explain the main differences now.

4.1 SAT instances

Much research has been done on meta-heuristics. Among them, Tabu search was introduced in 1986 by Glover [12] and extended to the SAT case in 1995 [25]. The main idea of tabu search consists, in a given position (interpretation), in exploring neighbours and choosing as the next position the one which minimises the objective function.

It is crucial to note that such an operation could increase the objective function value: it is the case when all neighbours have a greater value. Then, this mechanism allows to escape from local minimum. However, the main drawback is that at the next step, one goes back in the same local minimum. To avoid this, heuristic needs memory for the last explored positions to be forbidden. These positions are *tabu*.

Already explored positions are stored in a queue (usually called *tabu list*) of a given length which is a parameter of the method. This list must contain complete positions, which can be prohibitive. To go round this, one can store only previous actions, associated to values of the objective function. The length parameter is very important. A lot of work have been done to provide optimal length, statically [26] or dynamically [3].

We propose to keep the set of tabu variables by using a partial interpretation computed with unit propagation engine. When a variable becomes tabu, it is assigned in the CDCL solver part and propagated. Then, resulting interpretation is used as a tabu list. There are two advantages: firstly, the length of the tabu list is dynamic, it depends of unit propagation and backjumping. Secondly, unit propagation allows to catch some functional dependencies in the tabu list.

4.2 UNSAT instances

First of all, note that if an instance is unsatisfiable then, whatever is the complete interpretation, a falsified clause exists. Furthermore, if an instance is unsatisfiable, then it contains at least one MUS. This MUS, i.e. a subset of clauses of the formula, is often smaller than the global formula and, then, can contain less variables. Then, in the case of unsatisfiable formula, it is advantageous to focus the search on such variables.

In the frame of MUS detection, Grégoire *et al.* [13, 15] shown that local search provides good heuristics, concerning inconsistent kernel detection. These methods use properties 2 and 3 in order to balance clauses which could be part of a MUS.

The proposed method in this paper is based on this principle. When a local minimum is reached, property 2 assures that the set of clauses falsified by current interpretation are critical. Given that such clauses could be part of a MUS, we choose one of them to make it totally true. Therefore only the variables of a kernel are expected to be taken into account.

5 Implementation

As explicated in the previous section, the core of our solver SATHYS is the local search component. It is based on an iterative search process that in each step moves from one point to a neighbouring one until discovering a solution. At each step it tries to reduce the number of falsified clauses. When it is not possible, a local minimum is reached. In that case, the CDCL part is called. It chooses a falsified clause and assigns all of its literals such that the clause becomes totally valid. All literals of the chosen clause are decision nodes. Of course unit propagation is achieved. In this manner, it escapes from the local minimum and the SLS part of the hybrid solver can be used again. Note that all variables assigned by the CDCL part are fixed and can not be flipped by the SLS solver. Of course, during the CDCL process, a conflict can occur. In that case, conflict analysis is performed, a clause is learnt and a backjump is done. Then, some of fixed variables become free and can be flipped again. This conflict analysis makes the solver able to prove unsatisfiability.

Algorithm 2 takes a CNF formula Σ in parameter and returns SAT or UNSAT. It is based on WSAT-like algorithms. Two variables are used. A complete interpretation $\mathscr{I}_c$ for the local search engine (ini-

Algorithm 2: SATHYS

Input: Σ a CNF formula
Result: *SAT* if Σ is satisfiable, else *UNSAT*

```
1  while (true) do
2      I_c ← Init(Σ);
3      I_p ← ∅;
4      for j ← 1 to MaxFlips do
5          if I_c ⊨ Σ_{|I_p} then
6              return SAT;
7          Γ = {α ∈ Σ_{|I_p} | I_c ⊭ α}        /* set of falsified clauses */;
8          while Γ ≠ ∅ do
9              α ∈ Γ;
10             if ∃x ∈ α allowing a descent then
11                 flip(x);
12                 break;
13             else
14                 Γ ← Γ \ {α};
15         if Γ = ∅ then                        /* local minimum */
16             α ∈ Σ_{|I_p} such that I_c ⊭ α;
17             if (fix(Σ, I_c, I_p, α)=UNSAT) then
18                 return UNSAT;
```

tialised randomly) and a partial interpretation $\mathscr{I}_p$ for the CDCL part (initialised to the empty set). In order to forbid to flip fixed literals by the CDCL part (the literals of $\mathscr{I}_p$), the SLS solver deals with $\Sigma_{|\mathscr{I}_p}$. If the current complete interpretation is a model of $\Sigma_{|\mathscr{I}_p}$ then SATHYS finishes and returns SAT (lines 5–6). Otherwise, if it exists a neighbour of $\mathscr{I}_c$ which allows to decrease the number of falsified clauses, it becomes the current complete interpretation (lines 8–14). If it is not the case, then a local minimum is reached (line 15). In that case, a falsified clause is randomly chosen and the function `fix` is called in order to fix new literals (lines 15–17). This function is explained below. It modifies interpretations $\mathscr{I}_c$ and $\mathscr{I}_p$ by fixing new variables and (if a conflict occurs during boolean propagation) freeing other ones. At this step, the CDCL solver can prove the unsatisfiability. Of course, if it is the case the search is done (line 17–18).

This whole process is repeated a given number of times (*MaxFlips*, line 4). After that, the solver tries to go in another area of the search space. Then, the process can continue until finding an answer.

Function `fix` is described in Algorithm 3. It works like a very simple CDCL solver. It takes a clause α in input. It takes also in input the complete interpretation $\mathscr{I}_c$ and the partial one $\mathscr{I}_p$ and modifies them. It returns *UNSAT* if the unsatisfiability is proven, and *UNKNOWN* otherwise. The main goal of this function is to fix new variables. To achieve this, it tries to totally satisfy the clause α. First of all, the set of decision denoted $\mathscr{E}$ is initialized. Whenever it is not empty and a conflict does not occur, a new decision variable is chosen and added to the partial interpretation and boolean unit propagation (BCP) is performed (lines 3–6). If a conflict occurs, then the process is stopped. A conflict analysis is done and the partial interpretation is repaired (`backjumping`). At this step the unsatisfiability can be

Algorithm 3: fix

Input: α a clause
Output: Σ a CNF, $\mathscr{I}_c$ a complete interpretation, $\mathscr{I}_p$ a partial interpretation
Result: *UNSAT* if unsatisfiable is proven, *UNKNOWN* otherwise

1 $\gamma \leftarrow \emptyset$;
2 $\mathscr{E} \leftarrow \{x | \bar{x} \in \alpha\}$;
3 **while** $(\mathscr{E} \neq \emptyset)$ *and* $(\alpha = \emptyset)$ **do**
4 $\quad \mathscr{I}_p \leftarrow \mathscr{I}_p \cup \{x\}$ tel que $x \in \mathscr{E}$;
5 $\quad \gamma \leftarrow BCP()$;
6 $\quad \mathscr{E} \leftarrow \mathscr{E} \setminus \{x \in \mathscr{E} | \ x \in \mathscr{I}_p$ *or* $\bar{x} \in \mathscr{I}_p\}$;
7 **if** $\gamma \neq \emptyset$ **then**
8 $\quad \beta =$conflictAnalysis(Σ,$\mathscr{I}_p$,γ);
9 $\quad bl =$computeBackjumpingLevel(γ,$\mathscr{I}_p$);
10 $\quad$ **if** $(bl < 0)$ **then return** *UNSAT*;
11 $\quad \Sigma \leftarrow \Sigma \cup \{\beta\}$;
12 $\rho \leftarrow \{x \in \mathscr{I}_c | \ \bar{x} \in \mathscr{I}_p\}$;
13 $\mathscr{I}_c \leftarrow \mathscr{I}_c \setminus \{\bar{x} | \ x \in \rho\} \cup \rho$;
14 **return** *UNKONWN*;

proved. Otherwise, the obtained nogood is added to the clause database (lines 7–11). Then, the complete interpretation $\mathscr{I}_c$ is updated with the help of the partial one (note that $\mathscr{I}_p$ and $\mathscr{I}_c$ can not differ).

6 Experiments

Experimental results reported in this section were obtained on a Xeon 3.2 GHz with 2 GByte of RAM. The CPU time is limited to 1200 seconds.

Our approach is compared with:

- three SLS methods:
  1. classical WSAT [29], i.e. using random walk strategy
  2. RSAPS [21]
  3. ADAPTG2 [24]
- two recent hybrid methods submitted at the last SAT competition in 2009:
  1. HYBRIDGM [2]
  2. HYBRID1 [24]
- and two complete methods:
  1. CLS a local search method completed by adding resolution process [10]
  2. MINISAT [8] a well-known CDCL solver.

Instances used are taken from the last SAT competitions (`www.satcompetition.org`). They are divided into three different categories: crafted (1439 instances), industrial (1305) and random (2172). All instances are preprocessed with SatElite [7].

	Crafted		Industrial		Random	
	sat	unsat	sat	unsat	sat	unsat
ADAPTG2	326	0	232	0	1111	0
RSAPS	339	0	226	0	1071	0
WSAT	259	0	206	0	1012	0
CLS	235	75	227	102	690	0
SATHYS	322	191	466	309	341	14
HYBRIDGM	290	0	209	0	1114	0
HYBRID1	329	0	277	0	1126	0
MINISAT	402	369	588	414	609	315

Table 1: SATHYS versus some other SAT solvers

Table 1 summarizes the obtained results on this large number of instances. For more details on this experimental part, the reader can refer to `http://www.cril.fr/~lagniez/sathys`. For each category and for each solver we report the number of solved instances. Of course, MINISAT a state-of-the-art CDCL based complete solver is only considered to mention the gap between local search based techniques and complete modern SAT solvers on industrial and crafted instances. On random satisfiable instances, local search techniques generally outperform complete techniques.

Before analysing more precisely the table of results (Table 1), remark that only three solvers are able to solve unsatisfiable instances (MINISAT, CLS and SATHYS). The recent hybrid methods submitted at the last SAT competition cannot prove inconsistency in the allowed time.

On the crafted instances, SATHYS is very competitive and solves approximately the same number of satisfiable instances as RSAPS, ADAPTG2 and the recent hybrid methods. Furthermore, SATHYS solves much more instances than WSAT, its built-in solver. Concerning unsatisfiable crafted instances, as expected our approach is less efficient than MINISAT but it is proved highly more efficient than CLS.

Concerning industrial instances, SATHYS solves two times more satisfiable instances than SLS and hybrid methods. Once again, on unsatisfiable industrial instances, your solver is better than CLS but less efficient than MINISAT.

These results show that conflict analysis allows to solve efficiently structured SAT and UNSAT instances.

Finally, for the random category, we can note that SATHYS is unable to solve unsatisfiable problems. As pointed by MINISAT results, learning is not the good approach to solve random instances. As a summary, unfortunately our approach cannot reach the minisat performance. However the solver SATHYS is much more efficient than local search based algorithms and hybrid methods. It significantly improves WSAT, its built-in solver. Even if MINISAT is the best solver on crafted and industrial instances, these first results are very encouraging and reduce the gap between local search based techniques and DPLL-like complete solvers.

The figures 3, 4 and 5 give the classical cactus plot. For each tested method, the X-axis corresponds to the number of formulas and the Y-axis corresponds to the time needed to solve them if they were ran in parallel. When a method does not appear in the curve, that means that this method is not able to solve instance of this instances category. In these figures, we have distinguished satisfiable and unsatisfiable instances for each categories.

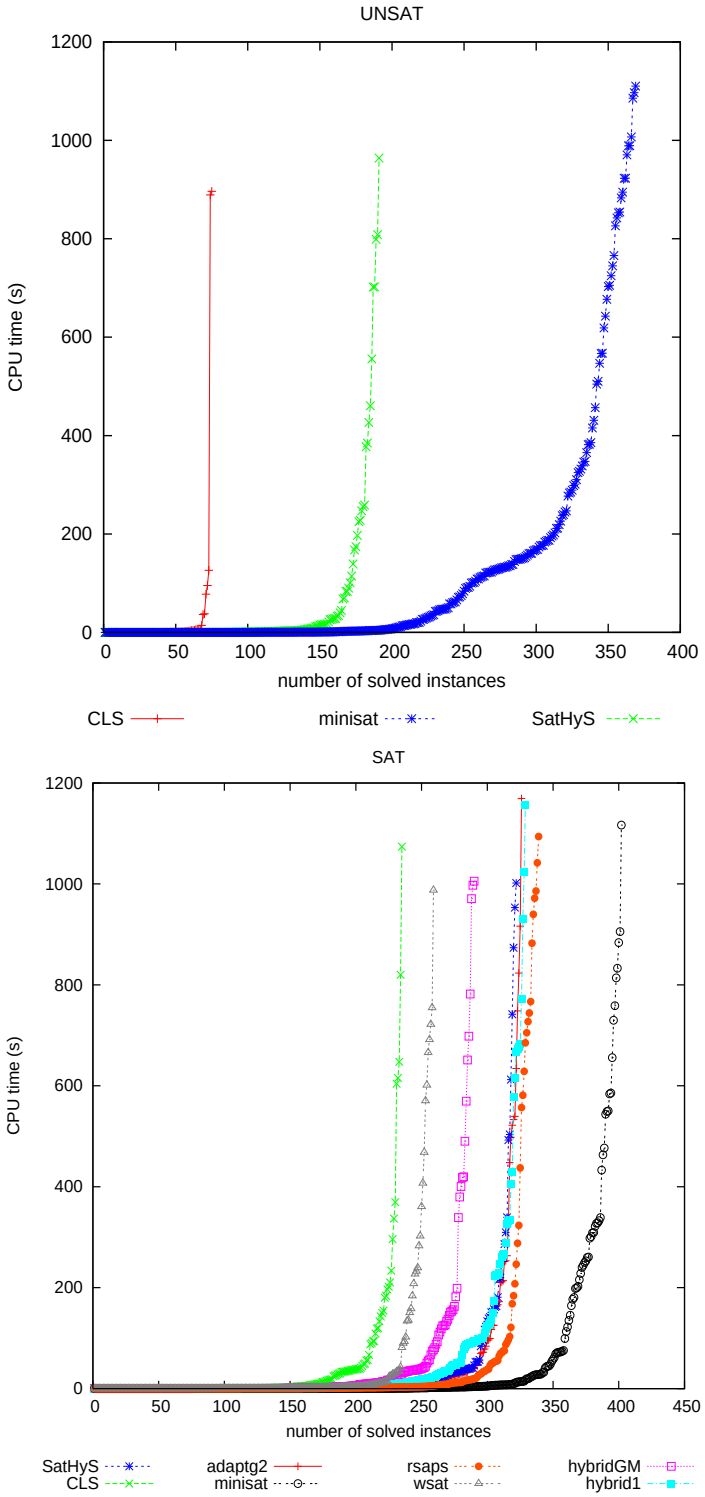


Figure 3: Crafted instances

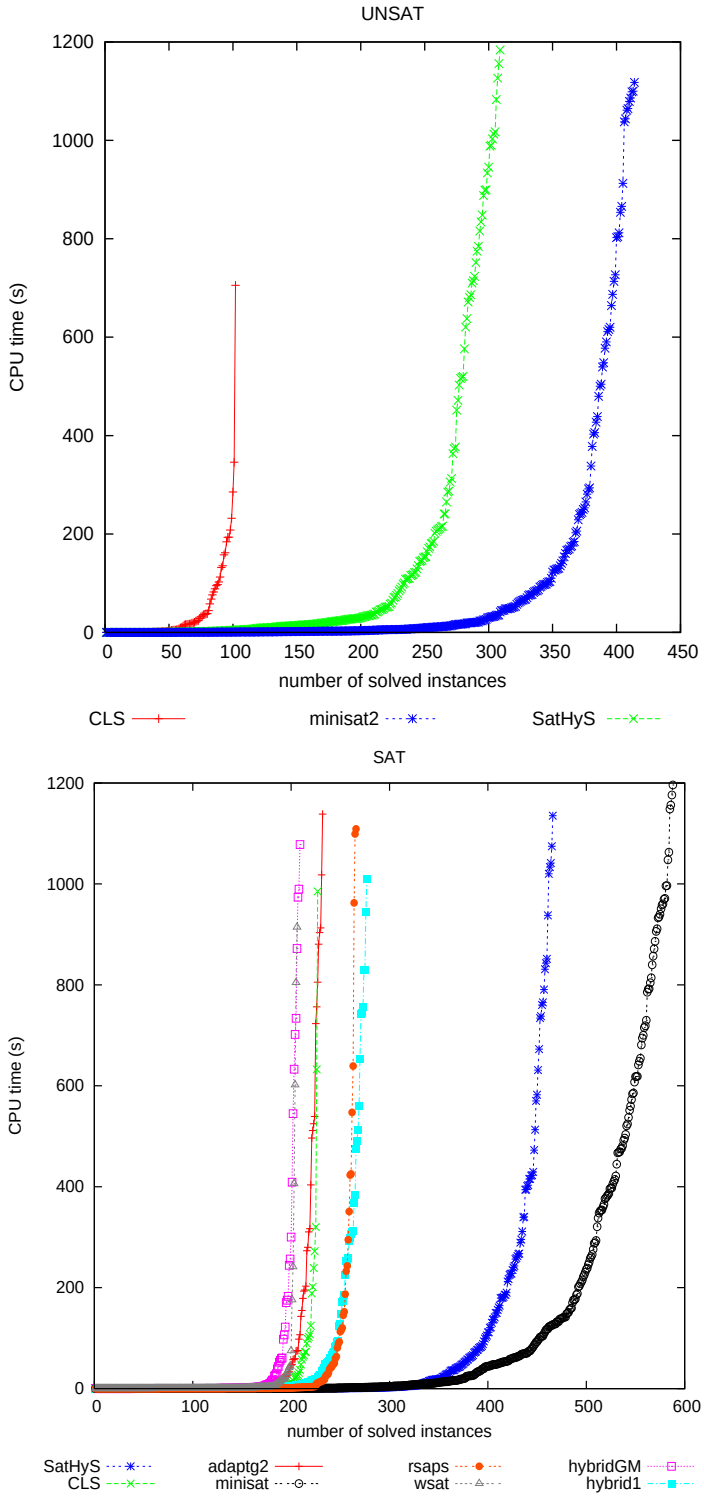


Figure 4: Industrial instances

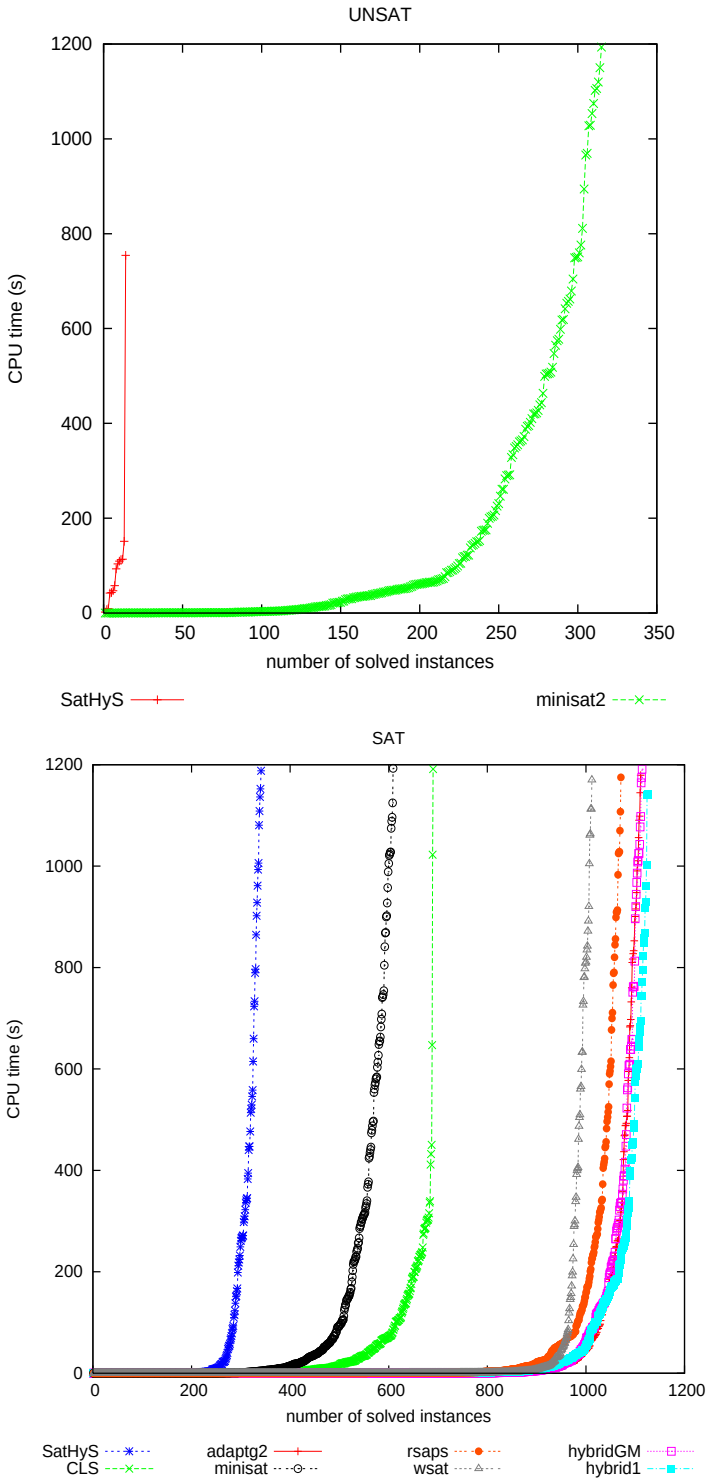


Figure 5: Random instances

7 Conclusion

In this paper a new integration of local search and CDCL based SAT solvers is introduced. This hybrid solver represents an original combination of both engines. The CDCL component can be seen as a new strategy for escaping from local minimum. This is achieved by the assignment of opposite literals from the falsified clause. In the case of satisfiable SAT instances, such assignments are supposed to behave like a tabu search approach, whereas for unsatisfiable ones, they try to focus on a small sub-part of the formula, which is minimally unsatisfiable (MUS). SATHYS, the resulting method, obtains very good results for a large category of instances. This new method can be improved in different ways. As it was pointed in the experimental section, our solver allows for more diversification and less intensification. First attempts have been done to correct this. Finally, we aim at designing a solver which would focuses only on an approximation of the MUS.

References

[1] B. B. & Fröhlich (2004): *WalkSAT as an Informed Heuristic to DPLL in SAT Solving.* Technical Report, CSE 573 : Artificial Intelligence.

[2] A. Balint, M. Henn & O. Gableske (2009): *A novel approach to combine a SLS- and DPLL-solver for the satisfiability problem*. In: *proceedings of SAT*.

[3] R. Battiti & M. Protasi (1997): *Reactive search, a history-sensitive heuristic for MAX-SAT*. *J. Exp. Algorithmics* 2, p. 2.

[4] A. Biere, M. Heule, H. van Maaren & T. Walsh, editors (2009): *Handbook of Satisfiability*, *Frontiers in Artificial Intelligence and Applications* 185. IOS Press.

[5] J. Crawford (1996): *Solving Satisfiability Problems Using a Combination of Systematic and Local Search*. In: *Second Challenge on Satisfiability Testing organized by Center for Discrete Mathematics and Computer Science of Rutgers University*.

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

[7] N. Eén & A. Biere (2005): *Effective Preprocessing in SAT Through Variable and Clause Elimination*. In: *proceedings of SAT*. pp. 61–75.

[8] N. Een & N. Sörensson (2003): *An Extensible SAT-solver*. In: *proceedings of SAT*. pp. 502–518.

[9] E.Gregoire, B. Mazure & C. Piette (2006): *Tracking MUSes and Strict Inconsistent Covers*. In: *Sixth ACM/IEEE International Conference on Formal Methods in Computer Aided Design(FMCAD'06)*. San Jose (USA), pp. 39–46.

[10] H. Fang & W. Ruml (2004): *Complete Local Search for Propositional Satisfiability*. In: *proceedings of AAAI*. pp. 161–166.

[11] L. Fang & M. Hsiao (2007): *A new hybrid solution to boost SAT solver performance*. In: *proceedings of DATE*. pp. 1307–1313.

[12] F. Glover (1989): *Tabu search - Part I*. *ORSA Journal of Computing* , pp. 190–206.

[13] E. Gregoire, B. Mazure & C. Piette (2006): *Extracting MUSes*. In: *proceedings of ECAI*. pp. 387–391.

[14] E. Gregoire, B. Mazure & C. Piette (2007): *Boosting a Complete Technique to Find MSS and MUS thanks to a Local Search Oracle*. In: *International Joint Conference on Artificial Intelligence(IJCAI'07)*. Hyderabad (India), pp. 2300–2305.

[15] E. Gregoire, B. Mazure & C. Piette (2007): *Local-Search Extraction of MUSes*. *Constraints* 12(3), pp. 325–344.

[16] D. Habet, C.M. Li, L. Devendeville & M. Vasquez (2002): *A Hybrid Approach for SAT*. In: *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*. pp. 172–184.

[17] W. Havens & B. Dilkina (2004): *A Hybrid Schema for Systematic Local Search*. In: *Canadian Conference on AI*. pp. 248–260.

[18] E. Hirsch & A. Kojevnikov (2005): *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination*. *Annals of Mathematical and Artificial Intelligence* 43(1), pp. 91–111.

[19] H. Hoos & T. Stützle (2004): *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier.

[20] J. Huang (2007): *The Effect of Restarts on the Efficiency of Clause Learning*. In: *proceedings of IJCAI*. pp. 2318–2323.

[21] F. Hutter, D. Tompkins & H. Hoos (2002): *Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT*. In: *proceedings of CP*. pp. 233–248.

[22] N. Jussien & O. Lhomme (2000): *Local Search with Constraint Propagation and Conflict-Based Heuristics*. In: *AAAI/IAAI*. pp. 169–174.

[23] F. Letombe & J. Marques-Silva (2008): *Improvements to Hybrid Incremental SAT Algorithms*. In: *proceedings of SAT*. pp. 168–181.

[24] C.M. Li, W. Wei & H. Zhang (2007): *Combining Adaptive Noise and Look-Ahead in Local Search for SAT*. In: *proceedings of SAT*. pp. 121–133.

[25] B. Mazure, L. Saïs & E. Grégoire (1995): *TWSAT : a new local search algorithm for SAT : performance and analysis*. In: *Proceedngs of the Workshop CP95 on Solving Really Hard Problems*. pp. 127–130.

[26] B. Mazure, L. Saïs & E. Grégoire (1998): *Boosting Complete Techniques Thanks to Local Search Methods*. *Ann. Math. Artif. Intell.* 22(3-4), pp. 319–331.

[27] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang & S. Malik (2001): *Chaff: Engineering an Efficient SAT Solver*. In: *proceedings of DAC*. pp. 530–535.

[28] B. Selman & H. Kautz (1993): *An Empirical Study of Greedy Local Search for Satisfiability Testing*. In: *proceedings of AAAI*. pp. 46–51.

[29] B. Selman, H. Kautz & B. Cohen (1994): *Noise Strategies for Improving Local Search*. In: *proceedings of AAAI*. pp. 337–343.

[30] B. Selman, H. Kautz & D. McAllester (1997): *Ten Challenges in Propositional Reasoning and Search*. In: *proceedings of IJCAI*. pp. 50–54.

[31] L. Zhang, C. Madigan, M. Moskewicz & S. Malik (2001): *Efficient Conflict Driven Learning in Boolean Satisfiability Solver*. In: *proceedings of ICCAD*. pp. 279–285.

Recovering and exploiting structural knowledge from CNF formulas

Article paru dans les actes de « *the Eighth International Conference on Principles and Practice of Constraint Programming* » (CP'02), lncs 2470, pages 185-199, septembre 2002.

Co-écrit avec Richard OSTROWSKI, Éric GRÉGOIRE et Lakhdar SAÏS.

Recovering and exploiting structural knowledge from CNF formulas

Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{ostrowski,gregoire,mazure,sais}@cril.univ-artois.fr

Abstract. In this paper, a new pre-processing step is proposed in the resolution of SAT instances, that recovers and exploits structural knowledge that is hidden in the CNF. It delivers an hybrid formula made of clauses together with a set of equations of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective operator among ($\vee$, $\wedge$, $\Leftrightarrow$) and where y and x_i are boolean variables of the initial SAT instance. This set of equations is then exploited to eliminate clauses and variables, while preserving satisfiability. These extraction and simplification techniques allowed us to implement a new SAT solver that proves to be the most efficient current one w.r.t. several important classes of instances.

Keywords: SAT, Boolean logic, propositional reasoning and search

1 Introduction

Recent impressive progress in the practical resolution of hard and large SAT instances allows real-world problems that are encoded in propositional clausal normal form (CNF) to be addressed (see e.g. [20, 10, 27]). While there remains a strong competition about building more efficient provers dedicated to hard random k-SAT instances [8], there is also a real surge of interest in implementing powerful systems that solve difficult large real-world SAT problems. Many benchmarks have been proposed and regular competitions (e.g. [6, 1, 22, 23]) are organized around these specific SAT instances, which are expected to encode structural knowledge, at least to some extent.

Clearly, encoding knowledge under the form of a conjunction of propositional clauses can flatten some structural knowledge that would be more apparent in a full propositional logic representation, and that could prove useful in the resolution step [21, 12].

In this paper, a new pre-processing step is proposed in the resolution of SAT instances, that extracts and exploits some structural knowledge that is hidden in the CNF. It delivers an hybrid formula made of clauses together with a set

of equations of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective operator among $\{\vee, \wedge, \Leftrightarrow\}$ and where y and x_i are Boolean variables of the initial SAT instance. Such an hybrid formula exhibits a twofold interest. On the one hand, the structural knowledge in the equations could be exploited by the SAT solver. On the other hand, these equations can allow us to determine equivalent variables and implied ones, in such a way that clauses and variables can be eliminated, while preserving satisfiability. These extraction and simplification techniques allowed us to implement a new SAT solver that proves to be the most efficient current one w.r.t. several important classes of instances.

The paper is organized as follows. After some preliminary definitions, it is shown how such a kind of equations can be extracted from the CNF, using a graph of clauses. Then, the task of simplifying the set of clauses using these equations is addressed. Experimental results showing the efficiency of the proposed approach are provided. Finally, promising paths of research are discussed in the conclusion.

2 Technical preliminaries

Let $\mathcal{L}$ be a Boolean (i.e. propositional) language of formulas built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) and a set of propositional variables. A *CNF formula* is a set (interpreted as a conjunction) of *clauses*, where a clause is a disjunction of *literals*. A literal is a positive or negated propositional variable. An *interpretation* of a Boolean formula is an assignment of truth values $\{true, false\}$ to its variables. A *model* of a formula is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist. Let c_1 be a clause containing a literal a and c_2 a clause containing the opposite literal $\neg a$, one *resolvent* of c_1 and c_2 is the disjunction of all literals of c_1 and c_2 less a and $\neg a$. A resolvent is called *tautological* when it contains opposite literals. Let us recall here that any Boolean formula can be translated thanks to a linear time algorithm in CNF, equivalent with respect to SAT (but that can use additional propositional variables). Most satisfiability checking algorithms operate on clauses, where the structural knowledge of the initial formulas is thus flattened.

Other useful definitions are the following ones. An *equation* or *gate* is of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective among $\{\vee, \wedge, \Leftrightarrow\}$ and where y and x_i are propositional variables. An equation is satisfied iff the left and right hand sides of the equation are simultanously $true$ or $false$. An interpretation of a set of equations is a model of this set iff it satisfies each equation of this set.

The first technical goal of this paper is to extract gates from a CNF formula. A propositional variable y (resp. $x_1, \ldots, x_n$) is an *output variable* (resp. are

input variables) of a gate of the form $y = f(x_1, \ldots, x_n)$. An output variable is also called *definable*.

A propositional variable z is an *output variable of a set of gates* iff z is an output variable of at least one gate in the set. An *input variable of a set of gates* is an input variable of a gate which is not an output variable of the set of gates.

Clearly, the truth-value of an y output variable depends on the truth value of the x_i input variables of its gate. Moreover, the set of definable variables of a CNF formula is a subset of the so-called *dependent* variables as defined in [15]. Knowing output variables can play an important role in solving the consistency status of a CNF formula. Indeed, the truth value of such variables can be obtained by propagation, and *e.g.* they can be omitted by selection heuristics of DPLL-like algorithms [4]. In the general case, knowing n' output variables of a CNF formula using n variables allows the size of the set of interpretations to be investigated to decrease from 2^n to $2^{n-n'}$.

Unfortunately, extracting gates from a CNF formula can be a time-consuming operation in the general case, unless some depth-limited search resources or heuristic criteria are provided. Indeed, showing that $y = f(x_1, \ldots, x_i)$ (where $y, x_1, \ldots, x_i$ belong to Σ), follows from a given CNF Σ, is coNP-complete [15].

3 Gates extraction

To the best of our knowledge, only equivalent gates were subject of previous investigation. Motivated by Selman et-al. challenge [24] about solving the parity-32 problems, Warners and van Maaren [26] have proposed an approach that succeeds in solving such class of hard CNF formulas. More precisely, a two steps algorithm is proposed: in the first one, a polynomially solvable subproblem (a set of equivalent gates) is identified thanks to a linear programming approach. Using the solution to this subproblem, the search-space is dramatically restricted for the second step of the algorithm, which is an extension of the well-known DPLL procedure [4]. More recently, Chu Min Li [18] proposed a specialised DPLL procedure called `EqSatz` which dynamically search for lists of equivalent literals, lists whose length is lower or equal to 3. Such an approach is costly as it performs many useless syntactical tests and suffers from restrictions (*e.g.* on the length of the detected lists).

In this paper, in order to detect hidden gates in the CNF formula, it is proposed to make use of an original concept of partial graph of clauses to limit the number of syntactical tests to be performed. Moreover, this technique allows gates $y = f(x_1, \ldots, x_n)$ (where $f \in \{\Leftrightarrow, \vee, \wedge\}$ and where no restriction on n is *a priori* given) to be detected.

Definition 1 (Graph of clauses)
Let Σ be a CNF formula. A **graph of clauses** *$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is associated to Σ s.t.*

4

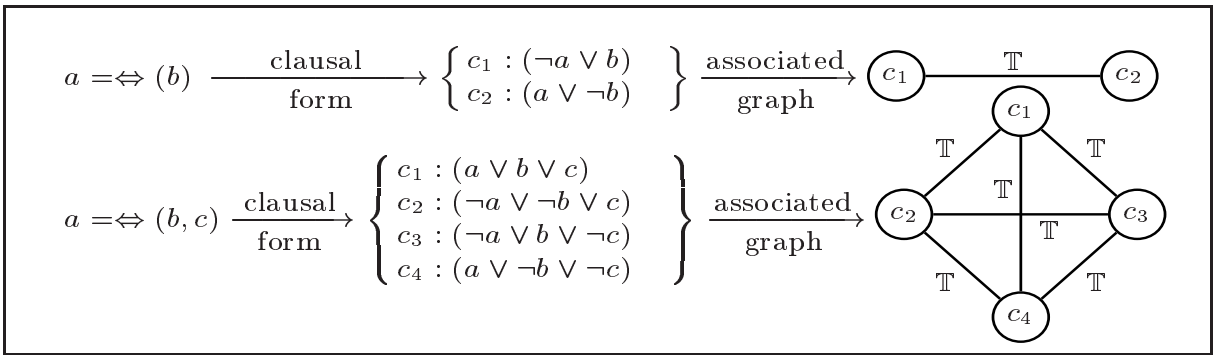
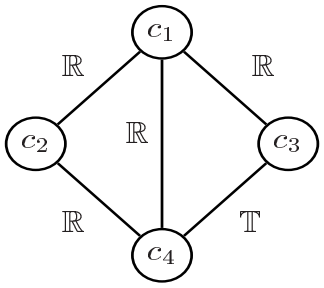


Fig. 1. Clausal and graphical representations of $\Leftrightarrow$ gates

- *each vertex of $\mathcal{V}$ corresponds to a clause of Σ;*
- *each edge (c_1, c_2) of $\mathcal{E}$ corresponds to a pair of clauses c_1 and c_2 of Σ exhibiting a resolvent clause;*
- *each edge is labeled either by $\mathbb{T}$ (when the resolvent is tautological) or $\mathbb{R}$ (when the resolvent is not tautological).*

Example 1 (Graph of clauses)

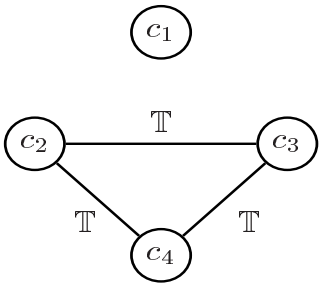
$$\Sigma = \left\{ \begin{array}{l} c_1 : (a \vee b \vee \neg c) \\ c_2 : (\neg a \vee b \vee d) \\ c_3 : (\neg a \vee b \vee \neg c) \\ c_4 : (a \vee c \vee \neg e) \end{array} \right\} \xrightarrow{\text{graph of clauses}} \mathcal{G} =$$



Definition 2 (Partial graph of clauses)
A **partial graph of clauses** *$\mathcal{G}'$ of a CNF formula Σ is the graph of clauses $\mathcal{G}$ of Σ that is restricted to edges labelled by $\mathbb{T}$.*

Example 2 (Partial graph of clauses)

$$\Sigma = \left\{ \begin{array}{l} c_1 : (a \vee d \vee e) \\ c_2 : (\neg a \vee \neg b \vee \neg c) \\ c_3 : (a \vee b \vee \neg c) \\ c_4 : (\neg a \vee b \vee c) \end{array} \right\} \xrightarrow[\text{of clauses}]{\text{partial graph}} \mathcal{G}' =$$



In Figure 1, both graphical and clausal representations of an equivalence gate $y = \Leftrightarrow (x_1, \ldots, x_n)$ ($n = 1$ and $n = 2$) are given. In the general case an equivalence gate will be represented by a partial graph that is a clique since any pair of clauses gives rise to a tautological resolvent.

In Figure 2, both graphical and clausal representations of gates $a = \wedge(b, c, d)$ and $a = \vee(b, c, d)$ are provided.

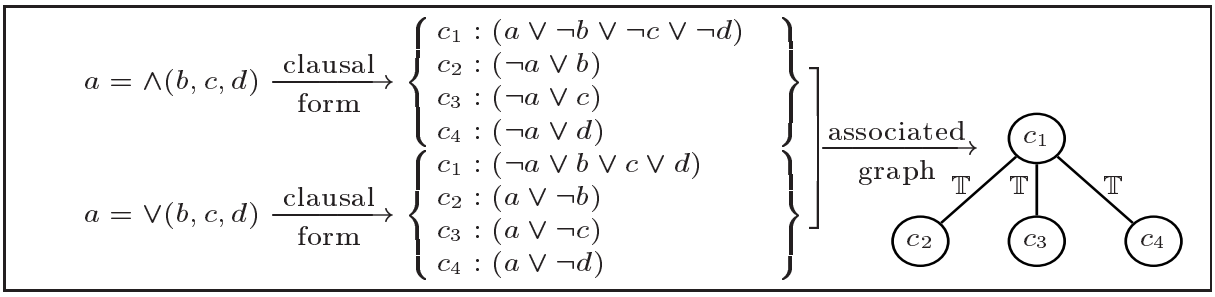


Fig. 2. Clausal and graphical representations of $\wedge$ and $\vee$ gates

Let us note that graphical representations of gates $\vee$ and $\wedge$ are identical since their clausal representations only differ by the variables signs. We also note that one clause plays a pivotal role and exhibits tautological resolvents with all clauses in the clausal representation of a $\vee$ or $\wedge$ gate. This property also applies for gates whose number of involved literals is greater than 3:

$$y = \vee(x_1, \ldots, x_n) \xrightarrow[\text{form}]{\text{clausal}} \left\{ \begin{array}{l} (\neg y \vee x_1 \cdots \vee x_n) \\ (y \vee \neg x_1) \\ \cdots \\ (y \vee \neg x_n) \end{array} \right\}$$

It is also easy to check that any resolvent from any pair of clauses from a same gate is tautological. Accordingly, the clauses from a same gate ($\Leftrightarrow$, $\vee$ and $\wedge$) are connected in the partial graph of clauses. Thus, a necessary (but not sufficient) condition for clauses to belong to a same gate is to form a connected subgraph. An example showing that such a condition is not sufficient is given in Example 2.

Building the graph of clauses is quadratic in the size of the set of clauses Σ but the representation of the graph can be too much space-consuming. Accordingly, finding gates will be performed in a dynamic manner (without representing the graph explicitly) by checking for each clause c which clauses of Σ exhibit tautological resolvents with c. This step can be achieved using two stacks:

- a stack of clauses sharing literals with c;
- a stack of clauses containing opposite literals to the literals of c.

At this step, the initial CNF formula is reduced to an hybrid formula, *i.e.* a set of equations together with initial clauses not taking part in these equations. For the uniformity of the representation each such remaining clause $c : (x_1 \vee \cdots \vee x_n)$ can be interpreted as a or gate of the form $true = \vee(x_1, \ldots, x_n)$ with its output variable assigned the value $true$ (but we shall sometimes still call them clauses, indifferently).

6

4 Exploiting structural knowledge

In this section, it is shown how such an obtained representation can even be simplified, and how its intrinsic properties can lead to more efficient satisfiability checking algorithms (by eliminating variables and clauses).

First of all, the n' output variables of the set of gates are clearly fixed by the remaining input variables. Accordingly, DPLL-like satisfiability algorithms [4] can restrict their search to input variables only.

Moreover, in the following it is shown how some properties of gates can allow even more variables and equations to be eliminated.

4.1 Equivalence gates

First, let us recall that when only equivalence gates are involved, then they can be solved in polynomial time [9]. These classes of formulas are known as chains of biconditionals. In the following, some basic properties of equivalence gates are presented. For more details, see, Dunham and Wang's paper [9]. For commodity, we use chains of biconditionals instead of equivalences gates.

Property 1 (about $\Leftrightarrow$ gates [9])

1. *$\Leftrightarrow$ is commutative and associative.*
2. *$(a \Leftrightarrow a \Leftrightarrow B)$ (resp. $(\neg a \Leftrightarrow a \Leftrightarrow B)$) with B a chain of biconditionnals is equivalent to B (resp $\neg B$).*
3. *$\neg(a \Leftrightarrow b \Leftrightarrow c)$ is equivalent to $(\neg a \Leftrightarrow b \Leftrightarrow c)$*
4. *$(\neg a \Leftrightarrow \neg b \Leftrightarrow \neg c)$ is equivalent to $(\neg a \Leftrightarrow b \Leftrightarrow c)$.*
5. *$(l \Leftrightarrow A_1), (l \Leftrightarrow A_2), \ldots, (l \Leftrightarrow A_m)$ is SAT iff $(A_1 \Leftrightarrow A_2), \ldots, (A_{m-1} \Leftrightarrow A_m)$ is SAT.*

It is easy to see that the first four equivalence gates properties apply on hybrid formulas.

As a consequence of the first property, any variable in an equivalence gate can play the role of the output variable of this gate. Currently, we have selected a very simple way for choosing output variables of equivalence gates. An output variable of an equivalence gate is selected among the set of output variables already defined for other gates. When the intersection of this set and the set of variables of the equivalence gate is empty, the output variable is selected in a random way in the set of variables involved in the equivalence gate. Properties 1.2. and 1.5. can lead to the elimination of variables and thus to a reduction of the search space. Property 1.4. shows that negation can be eliminated in pairs in chains of biconditionals (i.e. at most one literal of a chain of biconditionals is a negative one).

Let us now give new simplification properties of equivalent gates in the context of hybrid formulas (set of gates).

Property 2
Let Σ be a set of gates (i.e. an hybrid formula), $B \subset \Sigma$ a set of equivalence gates, $b \in B$ s.t. its output variable y occurs only in B and Σ' the set of gates obtained by the substitution of y with its definition and removing b from Σ, then Σ is satisfiable iff Σ' is satisfiable

Remark 1
The previous property is a simple extension of property 1.5 to set of gates.

Property 3
Let Σ be a set of gates, any equivalence gate of Σ containing a literal which does not occur elsewere in Σ, can be removed from Σ without loss of satisfiability.

Consequently, each literal in an equivalence gate must occur at least twice in the formula.

4.2 "And" & "or" gates

In the case of $\vee$ and $\wedge$ gates, the following property can be used to achieve useful simplifications.

Property 4 ($\vee$ and $\wedge$ gates)
- $a = f(b, c, b)$ *with* $f \in \{\vee, \wedge\}$ *is equivalent to* $a = f(b, c)$
- $a = \vee(b, c, \neg b)$ *(resp.* $a = \wedge(b, c, \neg b)$*) is equivalent to* a *(resp.* $\neg a$*)*
- $\neg a = \vee(b, c, d)$ *(resp.* $\neg a = \wedge(b, c, d)$*) is equivalent to* $a = \wedge(\neg b, \neg c, \neg d)$ *(resp.* $a = \vee(\neg b, \neg c, \neg d)$*)*
- *Property 2 and 3 hold for* $\vee$ *and* $\wedge$ *gates.*

4.3 Simplification of the remaining set of clauses

Let Γ be the remaining set of clauses (that can be interpreted as $true = \vee(x_1, \ldots, x_n)$ equations). Many practical approaches to SAT focus on the reduction of the number of variables of the instance to be solved, in order to reduce the size of the search space. In this paper, we also try to reduce the number of involved clauses. As we shall show it, this can lead to the elimination of variables, too. Interestingly, reducing the number of clauses and variables involved in Γ, leads to a reduction of the number of input variables.

In the following, two types of simplification are proposed. The first one is derived from an extension of the definition of blocked clauses, as proposed in [13, 14, 9]. The second one takes its roots in the pre-processing step of many efficient implementations of the DPLL algorithm [4]: the introduction of constant-length resolvents in a C-SAT-like spirit [7]. Let us note that, we can use other useful simplification techniques (see for example recent works by Brafman [3] and Marques-Silva [19]) to achieve further reduction on the CNF part of the formula.

8

Generalization of the blocked clause concept

Definition 3 (Blocked clause [13])
A clause c of a CNF formula Σ is **blocked** *iff there is a literal $l \in c$ s.t. for all $c' \in \Sigma$ with $\neg l \in c'$ the resolvent of c and c' is tautological.*

From a computational point of view, a useful property attached to the concept of blocked clause is the following one.

Property 5 (Blocked clause [13])
Let c be a clause belonging to a CNF formula Σ s.t. c is blocked. Σ is satisfiable iff $\Sigma \backslash \{c\}$ is satisfiable.

Example 3 (Blocked clause)
The following clause c_1 is blocked by the literal a.

$$\Sigma = \left\{ \begin{array}{l} c_1 : (a \vee b \vee c) \\ c_2 : (\neg a \vee \neg b) \\ c_3 : (\neg b \vee c) \\ c_4 : (b \vee \neg c) \end{array} \right\} \text{ is SAT iff } \Sigma \backslash \{c_1\} = \left\{ \begin{array}{l} c_2 : (\neg a \vee \neg b) \\ c_3 : (\neg b \vee c) \\ c_4 : (b \vee \neg c) \end{array} \right\} \text{ is SAT}$$

The concept of blocked clause can be generalized as follows, using the definition of non-fundamental clause.

Definition 4 (Non-fundamental clause)
A clause c belonging to a CNF formula Σ is **non-fundamental** *iff c is either tautological or is subsumed by another clause from Σ.*

From this, the concept of blocked clause is extended to *nf-blocked* clause.

Definition 5 (nf-blocked clause)
A clause c belonging to a CNF formula Σ is **nf-blocked** *iff there exists a literal l from c s.t. there does not exist any resolvent in l, or s.t. all resolvents are not fundamental.*

Property 5 can be extended to nf-blocked clauses.

Property 6 (nf-blocked clause)
Let c be a clause belonging to a CNF formula Σ s.t. c is nf-blocked. Σ is satifiable iff $\Sigma \backslash \{c\}$ is satisfiable.

Corollary 1
Blocked clauses and clauses containing a pure literal are nf-blocked.

The following example illustrates how the elimination of clauses can allow the consistency of a CNF formula to be proved.

Example 4 (nf-blocked)

$$\left\{\begin{array}{l} c_1 : (a \vee b \vee c) \\ c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{nf-blocked clause}]{c_1 \text{ nf-blocked by } a} \left\{\begin{array}{l} c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\}$$

$$\left\{\begin{array}{l} c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{blocked clause}]{c_2 \text{ nf-blocked by } b} \left\{\begin{array}{l} c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\}$$

$$\left\{\begin{array}{l} c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{blocked clause}]{c_3 \text{ nf-blocked by } d} \left\{\begin{array}{l} c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\}$$

$$\left\{\begin{array}{l} c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{pure literal}]{c_4 \text{ nf-blocked by } d} \left\{ c_5 : (a \vee b \vee \neg c) \right\}$$

$$\left\{ c_5 : (a \vee b \vee \neg c) \right\} \xrightarrow[\text{pure literal}]{c_5 \text{ nf-blocked by } a} \text{SAT}$$

As it can be done when pure literals are involved, this technique can lead to the elimination of variables. Indeed, it is always possible to nf-block a clause. To this end, we just have to add to the CNF all resolvents of the clause w.r.t. a given literal of this clause.

Property 7
Any clause c from a CNF formula Σ can be nf-blocked, introducing additional clauses in Σ.

In order to eliminate a variable, we just have to nf-block all clauses where it occurs. From a practical point of view, such a technique should be limited to variables giving rise to a minimum number of resolvents (*e.g.* variables which do not occur often). This idea is close to the elimination technique proposed in [5] and has been revisited in [25].

More generally, a concept of *redundant clause* can be defined as follows.

Definition 6 (Redundant clause [2]**)**
A clause c belonging to a CNF formula Σ is **redundant** *iff $\Sigma \backslash \{c\} \models c$.*

From a practical computational point of view, looking for redundant clauses amounts to proving that $\Sigma \wedge \neg c$ is inconsistent. Accordingly, it should not be searched for such clauses in the general case. However, it is possible to limit the search effort, *e.g.* by looking for implicates clauses or literals by unit propagation [16].

10

Definition 7 (u-redundant clause)
A clause c from a CNF formula Σ is **u-redundant** *iff the unsatisfiability of $\Sigma \wedge \neg c$ can be obtained using unit propagation, only (i.e. $\Sigma \backslash \{c\} \underset{Unit}{\models} c$).*

Clearly, the set of u-redundant clauses of a CNF formula is a subset of the set of redundant clauses of this formula. Using Example 4, the relationship between both nf-blocked and u-redandant clauses can be illustrated :

- nf-blocked clauses can be non u-redundant. (See clause c_1 in Example 4)
- u-redundant clauses can be non nf-blocked. (In the same example, if the initial CNF formula is extended with a clause $c_6 : (a \vee \neg d)$ and $c_7 : (\neg a \vee \neg d)$, then clause c_1 becomes u-redundant but is not nf-blocked anymore).
- Clauses can be u-redundant and nf-blocked at the same time. (In the same example, extending the initial CNF formula with both clauses $c'_6 : (b \vee c)$ and $c'_7 : (b \vee \neg c)$, clause c_1 remains nf-blocked and becomes u-redundant)

Limited form of resolution Many recent efficient implementations of DPLL [4] contain a preprocessing step introducing limited-length resolvents (the maximal length being generally fixed to 2), which increases the performance of the solver. However, the number of resolvents possibly introduced in this way can be prohibitive. Accordingly, we propose to limit the introduction of clauses to resolvents allowing clauses to be eliminated.

Definition 8 (Subsuming resolvent)
Let Σ be a CNF formula, a **subsuming resolvent** *is a resolvent from two clauses from Σ that subsumes at least one clause of Σ.*

Taking subsuming resolvents into account entails at least two direct useful consequences from a computational point of view. First, the subsuming resolvent is a shorter clause. Indeed, a subsuming clause is shorter than the subsumed one. From a practical point of view, we just need to eliminate one or some literals from the subsumed clause to get the subsuming one. Secondly, the elimination of such literals in the clause can lead to the suppression of a variable, or make it a unit literal or a pure literal. In all three cases, the search space is clearly reduced accordingly.

Example 5
Clauses $(a \vee b \vee c)$ and $(a \vee \neg c)$ generate the resolvent $(a \vee b)$, which subsumes the ternary clause and allows the literal c to be eliminated.

5 Implementation and experimental results

In this section, some preliminary -but significant- experimental results are presented. All algorithms have been programmed in `C` under Linux. All experimentations have been conducted using a 1 Ghz Pentium III processor, with 256 MB RAM, under Mandrake Linux 8.2.

instance	$\#C$	$\#V$	$\# \Leftrightarrow$	$\# \vee \wedge$	$\#C_\Gamma$	$\#V_\Gamma$	time(s)
par8-1-c	254	64	56	15	30	31	0.00
par8-1	1149	350	135	15	30	31	0.07
par16-1-c	1264	317	270	61	184	124	0.08
par16-1	3310	1015	560	61	184	124	0.25
par32-1-c	5254	1315	1158	186	622	375	0.38
par32-1	10277	3176	2261	186	622	375	0.64
barrel5	5383	1407	1065	152	1163	430	0.3
barrel6	8931	2306	1746	254	2013	821	0.53
barrel7	13765	3523	2667	394	3195	1337	0.96
barrel8	20083	5106	3864	578	4763	2158	1.80
ssa7552-125	3523	1512	1033	154	1270	501	0.33
ssa2670-130	3321	1359	859	254	1352	530	0.26
ssa0432-001	1027	435	225	43	244	124	0.1
bf1355-348	7271	2286	1082	383	3533	962	0.46
dubois100	800	300	200	0	0	0	0.05
2dlx_cc_mc_ex_bp_f2_bug091	55424	5259	0	4053	7575	5214	4.50
dlx1_c	1592	295	0	209	139	291	0.01
dlx2_cc_bug18	19868	2047	0	1567	1312	2039	0.92
dlx2_cc	12812	1516	0	1063	1137	1508	0.39
1dlx_c_mc_ex_bp_f	3725	776	0	542	378	755	0.05
2dlx_ca_mc_ex_bp_f	24640	3250	0	2418	1627	3223	0.94
2dlx_cc_mc_ex_bp_f	41704	4583	0	3534	2159	4538	2.88

Table 1. Number of extracted equations and time spent for the extraction

Before we implemented the solver, we addressed the *a priori* feasibility of the equations extraction technique, at least w.r.t. standard benchmarks. Indeed, although it is naturally expected that gates do exist in such benchmarks, these gates have never been exhibited. Moreover, despite the fact that the use of the partial graphs limits the number of syntactical tests to be performed, the extraction technique could appear too much time-consuming from a practical point of view.

The results given in Table 1 answer these issues for benchmarks from the last SAT competitions [6, 1, 22, 23]. For every tested instance, we have listed:

- the number of clauses ($\#C$) and of variables ($\#V$) of the initial instance;
- the number of discovered gates, using two separate categories: equivalence ($\# \Leftrightarrow$) and $\vee$ and $\wedge$ gates ($\# \vee \wedge$);
- the size of the set Γ of remaining clauses ($\#C_\Gamma$ & $\#V_\Gamma$);
- the time spent by the extraction process.

The results from Table 1 are really promising since they show that there exist many gates in many classes of benchmarks and that the time spent to find them is negligible (far less than 1 second, including the time spent to load the instance). Moreover, the size of the set Γ of remaining clauses after the extraction process is reduced in a significant manner (on average, the number of clauses is divided by a factor ranging from 2 to 10) and can even be zero for certain types of instances (*e.g.* `Dubois100`).

However, the set of variables from Γ and of the equations are not disjoint. We thus then focused on determining the number of variables that are really non defined, i.e. the variables that are never output ones. Table 2 provides the number of non defined variables (or input variables $\#V_{nd}$) for several instances,

12

instance	$\#C$	$\#V$	$\#V_\Gamma$	$\#V_{nd}$	$time(s)$
par8-1	1149	350	31	8	0.00
par16-1	3310	1015	124	16	0.05
par32-1	10277	3176	375	32	0.10
ssa0432-001	1027	437	106	63	0.00
ssa2670-140	3201	1327	444	196	0.01
ssa7552-001	3614	1534	408	246	0.02
bf2670-001	3434	1393	439	210	0.02
bf1355-160	7305	2297	866	526	0.06
bf0432-001	3668	1040	386	294	0.02
2dlx_cc_mc_ex_bp_f2_bug091	55424	5259	5214	1170	4.45
dlx1_c	1592	295	291	82	0.01
dlx2_cc_bug18	19868	2047	2039	477	0.92
dlx2_cc	12812	1516	1508	448	0.38
1dlx_c_mc_ex_bp_f	3725	776	755	214	0.0
2dlx_ca_mc_ex_bp_f	24640	3250	3223	807	0.97
2dlx_cc_mc_ex_bp_f	41704	4583	4538	1012	2.87

Table 2. Number of undefinable variables

notably for "parity" instances. These instances were selected because solving them is recognized as a challenge in the research community about SAT [24]. The results are quite surprising since only 32 variables are not defined w.r.t. the 3176 ones in the `par32-1` instance. This means that the truth value of the 3144 other variables depends only on these 32 variables obtained by the extraction technique. Accordingly, the search space is reduced from 2^{3176} to 2^{32} !

These abstraction and simplification techniques have been grafted as a pre-processing step to the DPLL procedure [4], using a branching heuristics *à la* Jeroslow-Wang [11]. This algorithm, called **LSAT** runs a DPLL-like algorithm on Γ and checks during the search process if the current interpretation being built does not contradict any detected gate. In the positive case, a backtrack step is performed. This new algorithm has been compared with the last versions of the most efficient SAT solvers, namely `Satz` [17], `EqSatz` [18], `Zchaff` [27]. The obtained results are given in Table 3 (time is given in seconds)[1].

These results show that LSAT is really more efficient that those solvers for many instances. Moreover, LSAT solves some instances in less than 1 second, whereas the other solvers took more than 16 minutes to give an answer.

6 Future work

This work opens promising paths for future research. Indeed, the current version of the LSAT solver is just a basic prototype that runs a DPLL procedure on the remaining clauses and checks that the current interpretation does not contradict the other equations. Clearly, such a basic prototype can be improved in several directions. First, it would be interesting to develop DPLL-specific branching heuristics that take all the equations into account (and not only the remaining clauses). It would also be interesting to explore how the algorithm could exploit

[1] In the table, $> n$ means that the instance could not be solved within n seconds.

instance	#C	#V	SAT	Satz	EqSatz	Zchaff	LSAT
par8-1	1149	350	yes	0.05	0.01	0.01	**0.01**
par8-2	1149	350	yes	0.04	0.01	0.01	**0.01**
par8-3	1149	350	yes	0.07	0.01	0.01	**0.01**
par8-4	1149	350	yes	0.09	0.01	0.01	**0.01**
par8-5	1149	350	yes	0.05	0.01	0.01	**0.01**
par16-1	3310	1015	yes	8.96	0.19	0.47	**0.05**
par16-2	3310	1015	yes	0.48	0.20	0.88	**0.05**
par16-3	3310	1015	yes	16.79	0.22	4.07	**0.02**
par16-4	3310	1015	yes	11.15	0.17	0.82	**0.06**
par16-5	3310	1015	yes	1.59	0.18	0.41	**0.03**
par32-1-c	5254	1315	yes	>1000	540	>1000	**6**
par32-2-c	5254	1315	yes	>1000	24	>1000	**28**
par32-3-c	5254	1315	yes	>1000	1891	>1000	**429**
par32-4-c	5254	1315	yes	>1000	377	>1000	**16**
par32-5-c	5254	1315	yes	>1000	4411	>1000	**401**
par32-1	10227	3176	yes	>1000	471	>1000	**27**
par32-2	10227	3176	yes	>1000	114	>1000	**7**
par32-3	10227	3176	yes	>1000	4237	>1000	**266**
par32-4	10227	3176	yes	>1000	394	>1000	**3**
par32-5	10227	3176	yes	>1000	5645	>1000	**471**
barrel5	5383	1407	no	86	0.38	1.67	**0.19**
barrel6	8931	2306	no	853	0.71	8.29	**0.55**
barrel7	13765	3523	no	>1000	0.96	21.55	**6.23**
barrel8	20083	5106	no	>1000	1.54	53.76	**412**
dubois10	80	30	no	0.03	0.08	0.01	**0.01**
dubois20	160	60	no	26.53	0.03	0.01	**0.01**
dubois30	240	90	no	>1000	0.05	0.01	**0.01**
dubois50	400	150	no	>1000	0.08	0.01	**0.01**
dubois100	800	300	no	>1000	0.06	0.06	**0.01**
Urquhart3	578	49	no	>1000	>1000	190	**0.02**
Urquhart4	764	81	no	>1000	>1000	>1000	**0.03**
Urquhart5	1172	125	no	>1000	>1000	>1000	**0.06**
Urquhart15	11514	1143	no	>1000	>1000	>1000	**0.42**
Urquhart20	18528	1985	no	>1000	>1000	>1000	**0.64**
Urquhart25	29670	3122	no	>1000	>1000	>1000	**1.05**

Table 3. Comparison of LSAT, Satz, EqSatz and Zchaff

the intrinsic properties of each type of equation.
In this paper, the simplification process of $\wedge$, $\vee$ gates and clauses has been described, but not yet implemented in the current LSAT version. On many classes of formulas (*e.g.* formal verification instances) containing a large part of such gates, we attempt further improvements using such simplification properties. More generally, it might be useful to extend this work to the simplification and resolution of general Boolean formulas. Finally, this work suggests that to model real-world problems, one might directly use more general and extended boolean formulas.

7 Conclusion

In this paper, a technique of extraction of equations of the form $y = f(x_1, \ldots, x_n)$ with $f \in \{\vee, \wedge, \Leftrightarrow\}$ from a CNF formula has been presented. This extraction technique allows us to rewrite the CNF formula under the form of a conjunction of equations. These equations classify variables into defined ones and undefined ones. The defined variables can be interpreted as the output of the logical gates

discovered by the extraction process, and allow us to reduce the search space in a significant way very often. Another contribution of this paper was the introduction of various simplification techniques of the remaining equations. In their turn, these latter techniques allow us to eliminate variables, reducing the search space again. These new techniques of extraction and simplification have been grafted as a pre-processing step of a new solver for SAT: namely, LSAT. This solver proves extremely competitive w.r.t. the best current techniques for several classes of structured benchmarks.

Acknowledgements

We are grateful to the anonymous referees for their comments on the previous version of this paper. This work has been supported in part by the CNRS, the "Conseil Régional du Nord/Pas-de-Calais", by the EC under a FEDER program, the "IUT de Lens" and the "Université d'Artois".

References

1. First international competition and symposium on satisfiability testing, March 1996. Beijing (China).
2. Yacine Boufkhad and Olivier Roussel. Redundancy in random sat formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000.
3. Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001.
4. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962.
5. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.
6. Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. `http://dimacs.rutgers.edu/Challenges/`.
7. Olivier Dubois, Pascal André, Yacine Boufkhad, and Jacques Carlier. Sat versus unsat. In D.S. Johnson and M.A. Trick, editors, *Second DIMACS Challenge*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, pages 415–436, 1996.
8. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, volume 1, pages 248–253, Seattle, Washington (USA), August 4–10 2001.
9. B. Dunham and H. Wang. Towards feasible solution of the tautology problem. *Annals of Mathematical Logic*, 10:117–154, 1976.
10. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, June 2001.

11. Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.
12. Henry A. Kautz, David McAllester, and Bart Selman. Exploiting variable dependency in local search. In *Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya (Japan), 1997.
13. Oliver Kullmann. Worst-case analysis, 3-sat decision and lower bounds: Approaches for improved sat algorithms. In *DIMACS Proceedings SAT Workshop*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, 1996.
14. Oliver Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, pages 1–72, 1997.
15. Jérome Lang and Pierre Marquis. Complexity results for independence and definability in propositional logic. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 356–367, Trento, 1998.
16. Daniel Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, Boston University, Massachusetts, USA, June 14th-15th 2001.
17. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya (Japan), August 1997.
18. C.M. Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 291–296, 2000.
19. Joao P. Marques-Silva. Algebraic simplification techniques for propositional satisfiability. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000)*, September 2000.
20. Shtrichman Oler. Tuning sat checkers for bounded model checking. In *Proceedings of Computer Aided Verification (CAV'00)*, 2000.
21. Antoine Rauzy, Lakhdar Saïs, and Laure Brisoux. Calcul propositionnel : vers une extension du formalisme. In *Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99)*, pages 189–198, Lyon, 1999.
22. Workshop on theory and applications of satisfiability testing, 2001. http://www.cs.washington.edu/homes/kautz/sat2001/.
23. Fifth international symposium on the theory and applications of satisfiability testing, May 2002. http://gauss.ececs.uc.edu/Conferences/SAT2002/.
24. Bart Selman, Henry A. Kautz, and David A. McAllester. Computational challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 50–54, Nagoya (Japan), August 1997.
25. A. Van Gelder. Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. *Annals of Mathematics and Artificial Intelligence*, 2002. to appear.
26. Joost P. Warners and Hans van Maaren. A two phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3–5):81–88, 1999.
27. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedigns of ICCAD'2001*, pages 279–285, San Jose, CA (USA), November 2001.

Automatic extraction of functional dependencies

Article sélectionné lors de « *the Eighth International Conference on Principles and Practice of Constraint Programming* » (SAT'04), pour parution dans lncs 3542, pages 122-132, 2005.

Co-écrit avec Éric GRÉGOIRE, Richard OSTROWSKI et Lakhdar SAÏS.

Automatic extraction of functional dependencies

Éric Grégoire, Richard Ostrowski, Bertrand Mazure, and Lakhdar Saïs

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{gregoire,ostrowski,mazure,sais}@cril.univ-artois.fr

Abstract. In this paper, a new polynomial time technique for extracting functional dependencies in Boolean formulas is proposed. It makes an original use of the well-known Boolean constraint propagation technique (BCP) in a new preprocessing approach that extracts more hidden Boolean functions and dependent variables than previously published approaches on many classes of instances.

Keywords: SAT, Boolean function, propositional reasoning and search.

1 Introduction

Recent impressive progress in the practical resolution of hard and large SAT instances allows real-world problems that are encoded in propositional clausal normal form (CNF) to be addressed (see e.g. [11, 7, 18]). While there remains a strong competition about building more efficient provers dedicated to hard random k-SAT instances [6], there is also a real surge of interest in implementing powerful systems that solve difficult large real-world SAT problems. Many benchmarks have been proposed and regular competitions (e.g. [4, 1, 14, 15]) are organized around these specific SAT instances, which are expected to encode structural knowledge, at least to some extent.

Clearly, encoding knowledge under the form of a conjunction of propositional clauses can flatten some structural knowledge that would be more apparent in more expressive propositional logic representation formalisms, and that could prove useful in the resolution step [13, 8].

In this paper, a new pre-processing step is proposed in the resolution of SAT instances, that extracts and exploits some structural knowledge that is hidden in the CNF. The technique makes an original use of the well-known Boolean constraint propagation (BCP) process. Whereas BCP is traditionally used to produce implied and/or equivalent literals, in this paper it is shown how it can be extended so that it delivers an hybrid formula made of clauses together with a set of equations of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective operator among $\{\vee, \wedge\}$ and where y and x_i are Boolean variables of the initial SAT instance. These Boolean functions allow us to detect a subset of dependent variables, that can be exploited by SAT solvers.

This paper extends in a significant way the preliminary results that were published in [12] in that it describes a technique that allows more dependent variables and hidden functional dependencies to be detected in several classes of instances. We shall see that the set of functional dependencies can underlie cycles. Unfortunately, highlighting actual dependent variables taking part in these cycles can be time-consuming since it coincides to the problem of finding a minimal cycle cutset of variables in a graph, which is a well-known NP-hard problem. Accordingly, efficient heuristics are explored to cut these cycles and deliver the so-called dependent variables.

The paper is organized as follows. After some preliminary definitions, Boolean gates and their properties are presented. It is then shown how more functional dependencies than [12] can be deduced from the CNF, using Boolean constraint propagation. Then, a technique allowing us to deliver a set of dependent variables is presented, allowing the search space to be reduced in an exponential way. Experimental results showing the interest of the proposed approach are provided. Finally, promising paths for future research are discussed in the conclusion.

2 Technical preliminaries

Let $\mathcal{B}$ be a Boolean (i.e. propositional) language of formulas built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) and a set of propositional variables.

A *CNF formula* Σ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive or negated propositional variable. We note $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) the set of variables (resp. literals) occurring in Σ. A *unit clause* is a clause formed with one unique literal. A *unit literal* is the unique literal of a unit clause.

In addition to these usual set-based notations, we define the negation of a set of literals ($\neg\{l_1, \ldots, l_n\}$) as the set of the corresponding opposite literals ($\{\neg l_1, \ldots, \neg l_n\}$).

An *interpretation* of a Boolean formula is an assignment of truth values $\{true, false\}$ to its variables. A *model* of a formula is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist.

Let c_1 be a clause containing a literal a and c_2 a clause containing the opposite literal $\neg a$, one *resolvent* of c_1 and c_2 is the disjunction of all literals of c_1 and c_2 less a and $\neg a$. A resolvent is called *tautological* when it contains opposite literals.

Let us recall here that any Boolean formula can be translated thanks to a linear time algorithm into CNF, equivalent with respect to SAT (but that can use additional propositional variables). Most satisfiability checking algorithms operate on clauses, where the structural knowledge of the initial formulas is thus flattened. In the following, CNF formulas will be represented as Boolean gates.

3 Boolean gates

A *(Boolean) gate* is an expression of the form $y = f(x_1, \ldots, x_n)$, where f is a standard connective among $\{\vee, \wedge, \Leftrightarrow\}$ and where y and x_i are propositional literals, that is defined as follows :

- $y = \wedge(x_1, \ldots, x_n)$ represents the set of clauses $\{y \vee \neg x_1 \vee \ldots \vee \neg x_n, \neg y \vee x_1, \ldots, \neg y \vee x_n\}$, translating the requirement that the truth value of y is determined by the conjunction of the truth values of x_i s.t. $i \in [1..n]$;
- $y = \vee(x_1, \ldots, x_n)$ represents the set of clauses $\{\neg y \vee x_1 \vee \ldots \vee x_n, y \vee \neg x_1, \ldots, y \vee \neg x_n\}$;
- $y =\Leftrightarrow (x_1, \ldots, x_n)$ represents the following *equivalence chain* (also called *biconditional formula*) $y \Leftrightarrow x_1 \Leftrightarrow \ldots \Leftrightarrow x_n$, which is equivalent to the set of clauses $\{y \vee x_1 \vee \ldots \vee x_n, y \vee \neg x_1 \vee \ldots \vee \neg x_n, \neg y \vee x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n, \ldots, \neg y \vee \neg x_1 \vee \ldots \vee \neg x_{n-1} \vee x_n\}$.

In the following, we consider gates of the form $y = f(x_1, \ldots, x_n)$ where y is a variable or the Boolean constant $true$, only.

Indeed, any clause can be represented as a gate of the form $true = \vee(x_1, \ldots, x_n)$. Moreover, a gate $\neg y = \wedge(x_1, \ldots, x_n)$ (resp. $\neg y = \vee(x_1, \ldots, x_n)$) is equivalent to $y = \vee(\neg x_1, \ldots, \neg x_n)$ (resp. $y = \wedge(\neg x_1, \ldots, \neg x_n)$). According to the well-known property of equivalence chain asserting that every equivalence chain with an odd (resp. even) number of negative literals is equivalent to the chain formed with the same literals, but all in positive (resp. except one) form, every gate of the form $y =\Leftrightarrow (x_1, \ldots, x_n)$ can always be rewritten into a gate where y is a positive literal. For example, $\neg y =\Leftrightarrow (\neg x_1, x_2, x_3)$ is equivalent to $y =\Leftrightarrow (x_1, x_2, x_3)$ and $\neg y =\Leftrightarrow (\neg x_1, x_2, \neg x_3)$ is equivalent to e.g. $y =\Leftrightarrow (x_1, x_2, \neg x_3)$.

A propositional variable y (resp. $x_1, \ldots, x_n$) is an *output variable* (resp. are *input variables*) of a gate of the form $y = f(x'_1, \ldots, x'_n)$, where $x'_i \in \{x_i, \neg x_i\}$.

A propositional variable z is an *output (dependent) variable of a set of gates* iff z is an output variable of at least one gate in the set. An *input (independent) variable of a set of gates* is an input variable of a gate which is not an output variable of the set of gates.

A gate is satisfied under a given Boolean interpretation iff the left and right hand sides of the gate are simultaneously $true$ or $false$ under this interpretation. An interpretation satisfies a set of gates iff each gate is satisfied under this interpretation. Such an interpretation is called a model of this set of gates.

4 From CNF to gates

Practically, we want to find a representation of a CNF Σ using gates that highlights a *maximal* number of dependent variables, in order to decrease the actual computational complexity of checking the satisfiability of Σ. Actually, we shall describe a technique that extracts gates that can be deduced from Σ, and that thus *cover* a subset of clauses of Σ. Remaining clauses of Σ will be represented as or-gates of the form $true = \vee(x_1, \ldots, x_n)$, in order to get a uniform representation.

More formally, assume that a set G of gates whose corresponding clauses $Cl(G)$ are logical consequences of a CNF Σ, the set $\Sigma_{uncovered(G)}$ of uncovered clauses of Σ w.r.t. G is the set of clauses of $\Sigma \backslash Cl(G)$.

Accordingly, $\Sigma \equiv \Sigma_{uncovered(G)} \cup Cl(G)$.

Not trivially, we shall see that the additional clauses $Cl(G) \backslash \Sigma$ can play an important role in further steps of deduction or satisfiability checking.

Knowing output variables can play an important role in solving the consistency status of a CNF formula. Indeed, the truth-value of an y output variable of a gate depends on the truth value of the corresponding x_i input variables. The truth value of such output variables can be obtained by propagation, and they can be omitted by selection heuristics of DPLL-like algorithms [3]. In the general case, knowing n' output variables of a gate-oriented representation of a CNF formula using n variables allows the size of the set of interpretations to be investigated to decrease from 2^n to $2^{n-n'}$. Obviously, the reduction in the search space increases with the number of detected dependent variables.

Unfortunately, to obtain such a reduction in the search space, one might need to address the following problems:

- Extracting gates from a CNF formula can be a time-consuming process in the general case, unless some depth-limited search resources or heuristic criteria are provided. Indeed, showing that $y = f(x_1, \ldots, x_i)$ (where $y, x_1, \ldots, x_i$ belong to Σ) follows from a given CNF Σ, is coNP-complete.
- when the set of detected gates contains recursive definitions (like $y = f(x, t)$ and $x = g(y, z)$), assigning truth values to the set of independent variables is not sufficient to determine the truth values of all the dependent ones. Handling such recursive definitions coincides to the well-known NP-hard problem of finding a minimal cycle cutset in a graph.

In this paper, these two computationally-heavy problems are addressed. The first one by restricting deduction to Boolean constraint propagation, only. The second one by using graph-oriented heuristics.

Let us first recall some necessary definitions about Boolean constraint propagation.

5 Boolean constraint propagation (BCP)

Boolean constraint propagation or unit resolution, is one of the most used and useful lookahead algorithm for SAT.

Let Σ be a CNF formula, $BCP(\Sigma)$ is the CNF formula obtained by *propagating* all unit literals of Σ. Propagating a unit literal l of Σ consists in suppressing all clauses c of Σ such that $l \in c$ and replacing all clauses c' of Σ such that $\neg l \in c'$ by $c' \backslash \{\neg l\}$. The CNF obtained in such a way is equivalent to Σ with respect to satisfiability.

The *set of propagated unit literals* of Σ using BCP is noted $UP(\Sigma)$. Obviously, we have that $\Sigma \vDash UP(\Sigma)$. BCP is a restricted form of resolution, and can be performed in linear time.

It is also complete for Horn formulas. In addition to its use in DPLL procedures, BCP is used in many SAT solvers as a processing step to deduce further interesting information such as implied [5] and equivalent literals [2][9]. Local processing based-BCP is also used to deliver promising branching variables (heuristic UP [10]).

In the sequel, it is shown that BCP can be further extended, allowing more general functional dependencies to be extracted.

6 BCP and functional dependencies

Actually, BCP can be used to detect hidden functional dependencies. The main result of the paper is the practical exploitation of the following original property: gates can be computed using BCP only, while checking whether a gate is a logical consequence of a CNF is coNP-complete in the general case.

Property 1. Let Σ be a CNF formula, $l \in \mathcal{L}(\Sigma)$, and $c \in \Sigma$ s.t. $l \in c$. If $c \backslash \{l\} \subset \neg UP(\Sigma \wedge l)$ then $\Sigma \models l = \wedge(\neg\{c \backslash \{l\}\})$.

Proof. Let $c = \{l, \neg l_1, \neg l_2, \ldots, \neg l_m\} \in \Sigma$ s.t. $c \backslash \{l\} = \{\neg l_1, \neg l_2, \ldots, \neg l_m\} \subset \neg UP(\Sigma \wedge l)$. The Boolean function $l = \wedge(\neg\{c \backslash \{l\}\})$ can be written as $l = \wedge(l_1, l_2, \ldots, l_m)$. To prove that $\Sigma \models l = \wedge(l_1, l_2, \ldots, l_m)$, we need to show that every model of Σ, is also a model of $l = \wedge(l_1, l_2, \ldots, l_m)$. Let I be a model of Σ, then

1. l is either *true* in I : I is also a model of $\Sigma \wedge l$. As $\{\neg l_1, \neg l_2, \ldots, \neg l_m\} \subset \neg UP(\Sigma \wedge l)$, we have $\{l_1, l_2, \ldots, l_m\} \subset UP(\Sigma \wedge l)$, then $\{l_1, l_2, \ldots, l_m\}$ are *true* in I. Consequently, I is also a model of $l = \wedge(l_1, l_2, \ldots, l_m\}\})$;
2. or l is *false* in I : as $c = \{l, \neg l_1, \neg l_2, \ldots, \neg l_m\} \in \Sigma$ then I satisfies $c = \{\neg l_1, \neg l_2, \ldots, \neg l_m\} \in \Sigma$. So, at least one the literals $l_i, i \in \{1, \ldots, m\}$ is *true* in I. Consequently, I is also a model of $l = \wedge(l_1, l_2, \ldots, l_m\}\})$

Clearly, depending on the sign of the literal l, and-gates or or-gates can be detected. For example, the and-gate $\neg l = \wedge(l_1, l_2, \ldots, l_n)$ is equivalent to the or-gate $l = \vee(\neg l_1, \neg l_2, \ldots, \neg l_n)$. Let us also note that this property covers binary equivalence since $a = \wedge(b)$ is equivalent to $a \Leftrightarrow b$.

Actually, this property allows gates to be detected, which were not in the scope the technique described in [12]. Let us illustrate this by means of an example.

Example 1. Let $\Sigma_1 \supseteq \{y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3, \neg y \vee x_1, \neg y \vee x_2, \neg y \vee x_3\}$.

According to [12], Σ_1 can be represented by a graph where each vertex represents a clause and where each edge corresponds to the existence of tautological resolvent between the two corresponding clauses. Each connected component might be a gate. As we can see the first four clauses belong to a same connected component. This is a necessary condition for such a subset of clauses to represent a gate. Such a restricted subset of clauses (namely, those appearing in the same connected component) is then checked syntactically to determine if it represents an and/or gate. Such a property can be checked in polynomial time. In the above example, we thus have $y = \wedge(x_1, x_2, x_3)$.

Now, let us consider, the following example,

Example 2. $\Sigma_2 \supseteq \{y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3, \neg y \vee x_1, \neg x_1 \vee x_4, \neg x_4 \vee x_2, \neg x_2 \vee x_5, \neg x_4 \vee \neg x_5 \vee x_3\}$.

Clearly, the graphical representation of this later example is different and the above technique does not help us in discovering the $y = \wedge(x_1, x_2, x_3)$ gate. Indeed, the above necessary but not sufficient condition is not satisfied.

Now, according to Property 1, both the and-gates behind Example 1 and Example 2 can be detected. Indeed, $UP(\Sigma_1 \wedge y) = \{x_1, x_2, x_3\}$ (resp. $UP(\Sigma_2 \wedge y) = \{x_1, x_4, x_2, x_5, x_3\}$) and

$\exists c \in \Sigma_1$, (resp. $c' \in \Sigma_2$), $c = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3)$ (resp. $c' = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3)$) such that $c \backslash \{y\} \subset \neg UP(\Sigma_1 \wedge y)$ (resp. $c' \backslash \{y\} \subset \neg UP(\Sigma_2 \wedge y)$).

Accordingly, a preprocessing technique to discover gates consists in checking the Property 1 for any literal occurring in Σ. A further step consists in finding dependent variables of the original formulas, as they can be recognised in the discovered gates. A gate clearly exhibits one dependent literal with respect to the inputs which are considered independent, as far a single gate is considered. Now, when several gates share literals, such a characterisation of dependent variables does not apply anymore. Indeed, forms of cycle can occur as shown in the following example.

Example 3. $\Sigma_3 \supseteq \{x = \wedge(y, z), y = \vee(x, \neg t)\}$.

Clearly, Σ_3 contain a cycle. Indeed, x depends on the variables y and z, whereas y depends on the variables x and t. When a single gate is considered, assigning truth values to input variables determines the truth value of the output, dependent, variable. As in Example 3, assigning truth values to input variables that are not output variables for other gates is not enough to determine the truth value of all involved variables. In the example, assigning truth values to z and t is not sufficient to determine the truth value of x and y. However, in the example, when we assign a truth value to an additional variable (x, which is called a *cycle cutset variable*) in the cycle, the truth value of y is determined. Accordingly, we need to cut such a form of cycle in order to determinate a sufficient subset of variables that determines the values of all variables. Such a set is called a *strong backdoor* in [17]. In Example 3, the strong backdoor corresponds to the set of $\{x\} \cup \{z, t\}$. In this context, a strong backdoor is the union of the set of independent variables and of the variables of the cycle cutset. Finding the minimal set of variables that cuts all the cycles in the set of gates is an NP-hard problem. This issue is investigated in the next section.

7 Searching for dependent variables

In the following, a graph representation of the interaction of gates is considered. More formally,

A set of gates can be represented by a bipartite graph $G = (O \cup I, E)$ as follows:

- for each gate we associate two vertices, the first one $o \in O$ represents the output of the gate, and the second one $i \in I$ represents the set of its input variables. So the number of vertex is less than $2 \times \#gates$, where $\#gates$ is the number of gates;
- For each gate, an edge (o, i) between the two vertices o and i representing the left and the right hand sides of a gate is created. Additional edges are created between $o \in O$ and $i \in I$ if one of the literals of the output variable associated to the vertex o belongs to the set of input literals associated to the vertex i.

Finding a *smallest* subset V' of O s.t. the subgraph $G' = (V' \cup O, E')$ is acyclic is a well-known NP-hard problem.

Actually, any subset V' that makes the graph acyclic is the representation of the set of variables, which together with all the independent ones, allows all variables to be determined. When V' is of size c, and the set of dependent variables is of size d, then the search space is reduced from 2^n to $2^{n-(d-c)}$, where n is the number of variable occurring in the original CNF formula.

We thus need to find a trade-off between the size of V', which influences the computational cost to find it, and the expected time gain in the subsequent SAT checking step.

In the following, two heuristics are investigated in order to find a cycle-cut set V'. The first-one is called *Maxdegree*. It consists in building V' incrementally by selecting vertices with the highest degree first, until the remaining subgraph becomes acyclic.

The second one is called *MaxdegreeCycle*. It consists in building V' incrementally by selecting first a vertex with the highest degree among the vertices that belong to a cycle. This heuristic guarantees that each time a vertex is selected, then at least one cycle is cut.

In the next section, extensive experimental results are presented and discussed, involving the preprocessing technique described above. It computes gates and cuts cycles when necessary in order to deliver a set of dependent variables. Two strategies are explored: in the first one, each time a gate is discovered, the covered clauses of Σ are suppressed; in the second one, covered clauses are eliminated at the end of the generation of gates, only. While the first one depends on the considered order of propagated literals, the second one is order-independent. These two strategies will be compared in terms of number of discovered gates, of the size of the cycle cutsets, of dependent variables and of the final uncovered clauses.

8 Experimental results

Our preprocessing software is written in C under Linux Redhat 7.1 (available at : `http://www.cril.univ-artois.fr/~ostrowski/Binaries/llsatpreproc`). All experimentations have been conducted on Pentium IV, 2.4 Ghz. Description of the benchmarks can be found on SATLib (http://www.satlib.org).

We have applied both [12] and our proposed technique on all benchmarks from the last SAT competition [15, 16], covering e.g. model-checking, VLSI and planning instances. Complete results are available at :
`http://www.cril.univ-artois.fr/~ostrowski/result-llsatpreproc.ps`. In the following, we illustrate some typical ones. On each class of instances, average and standard deviation results are provided with respect to the corresponding available instances.

In Table 1, for each considered class, the results of applying both [12]'s technique and the two new ones described above (in the first one, covered clauses are not suppressed as soon as they are discovered whereas they are suppressed in the second one) in terms of the mean number of discovered gates ($\#G$). The results clearly shows that our approach allows one to discover more gates. Not surprisingly, removing clauses conducts the number of detected gates to decrease.

Family of Instances		[12]'s technique	Our approach		
Name	(#Inst.,#V[min-Max],#C[min-Max])	#G	No cl. remov. #G	Cl. remov. #G	Cl. remov. #C remov.
Blocks	(3,484[283-758],27423[9690-47820])	10[3]	236[134]	18[5]	271[142]
Logistics	(8,994[116-3016],12706[953-50457])	380[265]	437[417]	169[213]	630[585]
Pipe	(6,1642[834-2577],18624[6695-33270])	1312[679]	1407[697]	1240[639]	13898[9083]
Facts	(13,3178[2218-4315],48737[22539-90646])	713[147]	1601[541]	497[170]	1731[510]
Parity	(30,1044[64-3176],3614[254-10325])	568[828]	510[594]	328[455]	663[870]
Qg	(10,969[512-1331],33747[9685-64054])	310[91]	1828[652]	298[80]	1708[601]
Ca	(7,637[26-2282],1835[70-6586])	419[547]	459[592]	414[542]	1233[1615]
Dp	(11,1427[213-3193],3580[376-8308])	1117[856]	1468[1211]	915[812]	2534[2298]
Bmc2	(5,1952[316-4089],6908[1002-13531])	895[714]	1025[850]	744[623]	2082[1824]
Rand	(6,2217[2000-2500],6568[5921-7401])	2133[236]	2444[381]	2103[252]	6212[692]
Ezfact	(40,1441[193-3073],9169[1113-19785])	40[18]	268[127]	68[33]	68[33]
Med	(3,761[341-1159],20154[5556-36291])	66[32]	316[162]	14[5]	319[164]
Avg-checker	(4,917[648-1188],28661[17087-40441])	324[105]	1098[375]	304[101]	1092[373]
nw/nc/fw	(13,3997[2756-5074],15829[10886-20123])	89[40]	468[136]	125[38]	125[38]
Am	(4,2011[433-4264],6925[1458-14751])	989[835]	772[585]	393[276]	927[625]
Cnf	(2,2424[2424-2424],14812[14812-14812])	2336[0]	3280[0]	2301[6]	13703[149]

Table 1. #G: Number of gates detected (average[standard deviation])

In Table 2, we took the no-remove option. We explored the above two heuristics for cutting cycles (*Maxdregre* and *MaxdegreeCycle*). For each class of instances, we provide the average number of detected dependent variables ($\#D$), the size of the cycle cutsets ($\#CS$) and the size of the discovered backdoor ($\#B$), and the cumulated CPU time in seconds for discovering gates and computing these results. On some classes, the backdoor can be 10% of the number of variables, only.

In Table 3, the remove option was considered. The number of gates is often lower than with the no-remove option. On the other hand, the size of the cycle cutset is generally lower with the remove option.

Accordingly, no option is preferable than the other one in the general case. Indeed, finding a smaller backdoor depends both on the considered class of instances and the considered option.

Family of Instances	(#V[min-Max])	*Maxdregre*			*MaxdegreeCycle*		
		#D	#CS	#B	#D	#CS	#B
Blocks	(484[283-758])	38[13]	198[123]	353[215]	39[9]	197[124]	352[216]
Logistics	(994[116-3016]	113[158]	245[218]	441[532]	143[164]	214[194]	410[522]
Pipe	(1642[834-2577])	980[768]	265[219]	582[201]	764[449]	481[192]	798[348]
Facts	(3178[2218-4315])	738[237]	813[256]	1964[604]	487[124]	1064[362]	2216[623]
Parity	(1044[64-3176])	243[388]	84[46]	573[528]	287[410]	40[21]	528[505]
Qg	(969[512-1331])	303[202]	228[236]	228[236]	11[6]	521[194]	521[194]
Ca	(637[26-2282])	290[434]	130[142]	344[403]	265[341]	155[206]	369[481]
Dp	(1427[213-3193])	513[463]	451[485]	725[625]	551[496]	412[343]	686[498]
Bmc2	(1952[316-4089])	662[716]	27[22]	886[874]	660[696]	30[10]	888[893]
Rand	(2217[2000-2500])	1777[301]	357[339]	440[343]	1152[134]	981[111]	1064[115]
Ezfact	(1441[193-3073])	28[35]	66[45]	1370[1073]	55[27]	39[18]	1343[1060]
Med	(761[341-1159])	205[102]	110[72]	110[72]	14[4]	302[157]	302[157]
Avg-checker	(917[648-1188])	209[357]	606[283]	606[283]	276[94]	539[187]	539[187]
nw/nc/fw	(3997[2756-5074])	39[48]	151[47]	3899[854]	94[24]	96[23]	3844[855]
Am	(2011[433-4264])	327[263]	97[68]	413[241]	298[206]	126[99]	441[287]
Cnf	(2424[2424-2424])	472[564]	1801[564]	1953[564]	1170[2]	1103[2]	1255[2]

Table 2. Size of backdoor with no remove option

Family of Instances	(#V[min-Max])	*Maxdegree*			*MaxdegreeCycle*		
		#D	#CS	#B	#D	#CS	#B
Blocks	(484[283-758])	18[4]	0[0]	373[219]	18[4]	0[0]	373[219]
Logistics	(994[116-3016]	135[147]	25[48]	419[539]	152[178]	7[13]	401[509]
Pipe	(1642[834-2577])	1020[735]	219[215]	543[223]	956[513]	282[124]	606[283]
Facts	(3178[2218-4315])	488[127]	0[0]	2214[621]	488[127]	0[0]	2214[621]
Parity	(1044[64-3176])	318[426]	0[0]	497[480]	318[426]	0[0]	497[480]
Qg	(969[512-1331])	122[99]	138[87]	410[189]	181[60]	80[25]	351[140]
Ca	(637[26-2282])	317[433]	94[113]	317[392]	302[388]	109[151]	332[434]
Dp	(1427[213-3193])	724[643]	149[151]	513[357]	728[641]	145[143]	509[353]
Bmc2	(1952[316-4089])	680[706]	1[1]	868[883]	680[705]	1[1]	868[884]
Rand	(2217[2000-2500])	1591[418]	495[396]	625[401]	1200[129]	886[102]	1016[111]
Ezfact	(1441[193-3073])	48[23]	10[5]	1350[1064]	49[23]	9[5]	1349[1064]
Med	(761[341-1159])	14[4]	0[0]	302[157]	14[4]	0[0]	302[157]
Avg-checker	(917[648-1188])	302[100]	0[0]	512[181]	302[100]	0[0]	512[181]
nw/nc/fw	(3997[2756-5074])	73[14]	40[22]	3864[857]	95[24]	18[10]	3842[856]
Am	(2011[433-4264])	367[254]	0[0]	373[239]	367[254]	0[0]	373[239]
Cnf	(2424[2424-2424])	1988[12]	285[12]	437[12]	2210[6]	63[6]	215[6]

Table 3. Size of backdoor with remove option

However, in most cases, the remove option and the *MaxdegreeCycle* heuristic lead to smaller backdoors.

We are currently experimenting how such a promising preprocessing step can be grafted to the most efficient SAT solvers, allowing them to focus directly on the critical variables of the instances (i.e. the backdoor). Let us stress that our preprocessing step has been implemented in a non-optimized way. However, it shows really viable thanks to good obtained computing time (**less than 1 second in most cases**), so time is omitted in different tables.

9 Future works

Let us here simply motivate another interesting path for future research, related to the actual expressiveness of discovered clauses. Actually, our gate-oriented representation of a Boolean formula exhibits additional information that can prove powerful with respect to further steps of deduction or satisfiability checking. To illustrate this, let us consider Example 2 again. From the CNF Σ, the gate $y = \wedge(x_1, x_2, x_3)$ is extracted. The clausal representation of the gate is given by $\{y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3, \neg y \vee x_1, \neg y \vee x_2, \neg y \vee x_3\}$.

Clearly, the additional clauses $\{\neg y \vee x_2, \neg y \vee x_3\}$ are resolvents from Σ, which can only be obtained using two and six basic steps of resolution, respectively. Accordingly, the gate representation of Σ involves non-trivial binary resolvents, which can ease further deduction or satisfiability checking steps. Taking this feature into account either in clausal-based or gate-based deduction of satisfiability solvers should be a promising path for future research. Also, some of the discovered gates represent equivalencies ($x \Leftrightarrow y$), substituting equivalent literals might lead to further reductions with respect to the number of variables.

Another interesting path for future research concerns the analysis of the obtained graph and the use of e.g. decomposition techniques. To further reduce the size of the backdoor, we also plan to study how tractable parts of the formula (e.g. horn or horn-renommable) can be exploited.

10 Conclusions

Clearly, our experimentations results are encouraging. Dependent variables can be detected in a preprocessing step at a very low cost. Cycles occur, and they can be cut. We are currently grafting such a preprocessing technique to efficient SAT solvers. Our preliminary experimentations show that this proves often beneficial. Moreover, we believe that the study of cycles and of dependent variables can be essential in the understanding of the difficulty of hard SAT instances.

11 Acknowledgements

This work has been supported in part by the CNRS, the FEDER, the *IUT de Lens* and the *Conseil Régional du Nord/Pas-de-Calais*.

References

1. First international competition and symposium on satisfiability testing, March 1996. Beijing (China).
2. L. Brisoux, L. Sais, and E. Grégoire. Recherche locale : vers une exploitation des propriétés structurelles. In *Actes des Sixièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets(JNPC'00)*, pages 243–244, Marseille, 2000.
3. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962.
4. Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. `http://dimacs.rutgers.edu/Challenges/`.
5. Olivier Dubois, Pascal André, Yacine Boufkhad, and Jacques Carlier. Sat versus unsat. In D.S. Johnson and M.A. Trick, editors, *Second DIMACS Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, pages 415–436, 1996.
6. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, volume 1, pages 248–253, Seattle, Washington (USA), August 4–10 2001.
7. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, June 2001.
8. Henry A. Kautz, David McAllester, and Bart Selman. Exploiting variable dependency in local search. In *Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya (Japan), 1997.
9. Daniel Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, Boston University, Massachusetts, USA, June 14th-15th 2001.
10. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya (Japan), August 1997.
11. Shtrichman Oler. Tuning sat checkers for bounded model checking. In *Proceedings of Computer Aided Verification (CAV'00)*, 2000.
12. Grégoire E. Mazure B. Ostrowski R. and Sais L. Recovering and exploiting structural knowledge from cnf formulas. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, pages 185–199, Ithaca (N.Y.), 2002. LNCS 2470, Springer Verlag.
13. Antoine Rauzy, Lakhdar Saïs, and Laure Brisoux. Calcul propositionnel : vers une extension du formalisme. In *Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99)*, pages 189–198, Lyon, 1999.
14. Sat 2001: Workshop on theory and applications of satisfiability testing, 2001. http://www.cs.washington.edu/homes/kautz/sat2001/.
15. Sat 2002 : Fifth international symposium on theory and applications of satisfiability testing, May 2002. http://gauss.ececs.uc.edu/Conferences/SAT2002/.
16. Sat 2003 : Sixth international symposium on theory and applications of satisfiability testing, May 2003. http://www.mrg.dist.unige.it/events/sat03/.
17. Ryan Williams, Carla P. Gomez, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178, 2003.
18. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedigns of ICCAD'2001*, pages 279–285, San Jose, CA (USA), November 2001.

Reducing hard SAT instances to polynomial ones

Article publié dans « *the 2007 IEEE international conference on Information Reuse and Integration* » (IEEE-IRI'07), pages 18-23, Las Vegas, Nevada, USA, août 2007.

Co-écrit avec Olivier FOURDRINOY, Éric GRÉGOIRE et Lakhdar SAÏS.

Reducing hard SAT instances to polynomial ones

Olivier Fourdrinoy Éric Grégoire Bertrand Mazure Lakhdar Saïs
CRIL CNRS & IRCICA
Université d'Artois
Rue Jean Souvraz SP18
F-62307 Lens Cedex France
{fourdrinoy, gregoire, mazure, sais}@cril.fr

Abstract

This last decade, propositional reasoning and search has been one of the hottest topics of research in the A.I. community, as the Boolean framework has been recognized as a powerful setting for many reasoning paradigms thanks to dramatic improvements of the efficiency of satisfiability checking procedures. SAT, namely checking whether a set of propositional clauses is satisfiable or not, is the technical core of this framework. In the paper, a new linear-time pre-treatment of SAT instances is introduced. Interestingly, it allows us to discover a new polynomial-time fragment of SAT that can be recognized in linear-time, and show that some benchmarks from international SAT competitions that were believed to be difficult ones, are actually polynomial-time and thus easy-to-solve ones.

1 Introduction

These last decade, propositional reasoning and search has been one of the hottest topics of research in the A.I. community, as the Boolean framework has been recognized as a powerful setting for many reasoning paradigms thanks to dramatic improvements of the efficiency of satisfiability checking procedures. SAT, namely checking whether a set of propositional clauses is satisfiable or not, is the technical core of this framework. In the paper, a new linear-time pre-treatment of SAT instances is introduced. Interestingly, it allows us to discover a new polynomial-time fragment of SAT that can be recognized in linear-time, and show that some benchmarks from international SAT competitions that were believed to be difficult ones, are actually polynomial-time and thus easy-to-solve ones. Many approaches have been proposed to solve hard SAT instances. Direct approaches have focused on the development of -logically complete or not- algorithms. Local-search techniques (e.g. [21]) and elaborate variants of the Davis-Loveland-Logemann's DPLL procedure [8] (e.g. [18, 12]) allow many families of difficult instances to be solved. Indirect approaches aim at solving instances, using either approximation or compilation techniques (see e.g. [7, 3, 20]). In particular, compilation techniques, which were developed in the more general framework of propositional deduction, aim at transforming the set of Boolean clauses into a deductively equivalent form that belongs to a polynomial fragment, making use of a -possibly exponential- transformation schema and by ensuring that the compiled form remains tractable in size. Finally, other approaches have concentrated on discovering and studying fragments of SAT that can be recognized and solved in polynomial time (see e.g. [10, 5, 2, 4]). The contribution of this paper pertains to these three families of approaches. A new pre-treatment of SAT instances is introduced: it can be performed before some direct approaches are run. It can be interpreted as an attempt to compile the SAT instance into an easier-to-solve one. However, contrary to usual compilation techniques, the transformation process remains a polynomial-time one, and no guarantee is provided that the resulting set of clauses belongs to a polynomial fragment. However, this pre-treatment can prove valuable in showing that some instances are actually polynomial ones or in making the further solving step become more efficient. Finally, a new polynomial fragment of SAT, called U-Horn SAT (*Horn modulo Unit propagation*), is put in light. Interestingly, it can be recognized using the proposed polynomial-time pre-treatment. In other words, SAT instances that can be mapped to Horn SAT using our approach belong to the U-Horn SAT fragment. Roughly, this pre-treatment is as follows. The focus is on the unit propagation mechanism (in short UP), which is a linear-time deductive mechanism. Given a polynomial fragment (e.g. the Horn one), any SAT instance can be divided into two subsets of clauses: the first one contains clauses that belong to the targeted polynomial fragment whereas the second cone contains clauses that do not belong to it. For each of these latter clauses, we at-

tempt to discover one sub-clause belonging to the polynomial fragment, using UP. In case of success, this sub-clause can replace the initial one, and increase the size of the polynomial subset. In case of failure, we also check whether the clause itself is an UP consequence of the instance or not. The paper is organized as follows. In the next section, the basic formal background is provided, together with the description of propositional fragments that will be mentioned in the paper. Then, the pre-treatment is described, before extensive experimental studies are reported and analyzed. Then, it is shown how this approach extends some previous related works and other SAT-related approaches exploiting the unit propagation mechanism.

2 Technical background

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables, noted a, b, c, etc. Formulas will be noted using upper-case letters such as C. Sets of formulas will be represented using Greek letters like Γ or Σ. An interpretation is a truth assignment function that assigns values from {*true, false*} to every Boolean variable. A formula is consistent or satisfiable when there is at least one interpretation that satisfies it, i.e. that makes it become *true*. An interpretation will be noted by upper-case letters like I and will be represented by the set of literals that it satisfies. Actually, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) using a set (interpreted as a conjunction) of clauses, where a clause is a finite disjunction of literals, where a literal is Boolean variable that can negated. Clauses will be represented by the set of literals that they contain. For example, the clause $C = a \vee b \vee \neg c \vee \neg d$ will be represented by the set $\{a, b, \neg c, \neg d\}$. A clause is said to be positive (resp. negative) if it contains no negative (resp. positive) literal. The size of a clause is the number of literals in it. Unit clauses contain exactly one literal whereas binary ones contain at most two literals. The empty clause is denoted by $\bot$. A clause C is a sub-clause of a clause D iff $C \subseteq D$. For example, the resolvent of $C_1 = (p \vee \alpha)$ and $C_2 = (\neg p \vee \beta)$ is defined as $Res(C_1, C_2) = (\alpha \vee \beta)$ (Resolution rule); it is a logical consequence of C_1 and C_2. SAT is the NP-complete problem that consists in checking whether a set of Boolean clauses (also called CNF) is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not. A central deductive mechanism in this paper is the unit propagation mechanism (in short UP). UP is a linear time process that recursively simplifies a SAT instance by propagating the constraints expressed by unit clauses. Let Σ be a SAT instance, $UP(\Sigma)$ is defined as the formula obtained by unit propagation. A clause C is a UP consequence of Σ; noted $\Sigma \models^* C$, iff $UP(\Sigma \wedge \neg C)$ allows to derive the empty clause. A clause C' is called a sub-clause of C if $C' \subset C$. A sub-clause C' of C is called maximal if $|C| - |C'| = 1$. Some fragments of $\mathcal{L}$ exhibit polynomial-time algorithms for SAT. Among them, let us mention the Horn fragment, which is made of Horn clauses only. A Horn (resp. reverse Horn) clause contains at most one positive (resp. negative) literal. Binary and renamable Horn clauses also form polynomial fragments: renamable Horn clauses are clauses that can be transformed into Horn ones by systematically replacing some negative literals by new Boolean variables. Let us also mention Dalal and Etherington's hierarchy of classes [5] and the class of Q-Horn [2] formulas, which strictly contains all binary, Horn reverse, and renamable-Horn clauses. All of them can be recognized and solved in polynomial time. A polynomial fragment of $\mathcal{L}$ of special interest in this paper is Quad, introduced by Dalal [4]. Quad is based on a tractable fragment called $Root$. A formula Σ is in class $Root$, if either (1) Σ contains the empty clause, or (2) Σ contains no positive clause, or (3) Σ contains no negative clause, or (4) all clauses of Σ are binary. A formula Σ is in class Quad[4] if either (1) $UP(\Sigma)$ belongs to Root, or (2) for the first max sub-clause C' of the first clause $C \in UP(\Sigma)$ for which $UP(\Sigma \wedge \neg C')$ is in class Root: (a) either $UP(\Sigma \wedge \neg C')$ is unsatisfiable, or (b) the formula $(\Sigma \setminus \{C\}) \cup \{C'\}$ is in class Quad. As mentioned by Dalal, Quad depends on the considered ordering of clauses. Different orderings might lead to different Quad classes.

3 A new pre-treatment

The central idea is to reduce the non-polynomial fragment of the SAT instance Σ through the use of UP. Σ can be divided in two parts. The first one is formed by polynomial-time detectable clauses w.r.t. a given polynomial fragment, like Horn, reverse-Horn or strictly positive formulas, etc. The second one contains the remaining clauses. When the second part is empty, Σ is polynomial. Different forms of sub-clause deduction can be defined depending on the targeted polynomial class. For example, if binary clauses are considered then sub-clause deduction of binary clauses will concern clauses whose length is larger than two. In the following, we instantiate this general approach by selecting the Horn fragment as the polynomial target. First, let us introduce some necessary definitions.

Definition 1 (U-Horn clause)
Let $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$, with $n \geq 0$ and $p > 1$ a clause of Σ. C is called a U-Horn clause of Σ iff $\exists C' = \{\neg n_1, \ldots, \neg n_n, p_i\}$ a sub-clause of C, s.t. $\Sigma \models^* C'$ or $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$.

Property 1 *If $C \in \Sigma$ is a U-Horn clause and $C' \subset C$ is a Horn clause s.t. $\Sigma \models^* C'$ then Σ is satisfiable if and only if $(\Sigma \setminus \{C\}) \cup \{C'\}$ is satisfiable.*

The above property states that when a clause C of Σ is U-Horn, it can be replaced in Σ by a Horn clause without any change in the satisfiability status of Σ.

Definition 2 (U-redundant [13])
A clause C of Σ is called U-redundant iff $\Sigma \setminus \{C\} \models^* C$.

Property 2 *If $C \in \Sigma$ is a U-redundant clause then Σ is satisfiable iff $\Sigma \setminus \{C\}$ is satisfiable.*

Thus, U-redundant clauses can be safely removed from Σ. Let us now introduce two properties, leading to the introduction of two new additional reduction operators, namely $U\text{-}NRes$ and $U\text{-}PRes$, respectively.

Property 3 *Let $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ be a clause of Σ. If $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n\}$ or $\exists p_i \in C$ s.t. $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ and $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$, then $\Sigma \models \{\neg n_1, \ldots, \neg n_n\}$.*

Definition 3 (U-NRes)
When a clause $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ of Σ satisfies Property 3, $U\text{-}NRes(C)$ is defined as $\{\neg n_1, \ldots, \neg n_n\}$.

Property 4 *Let $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ be a clause of Σ. If $\exists p_i \in C$ s.t. $\Sigma \not\models^* \{\neg n_1, \ldots, \neg n_n, p_i\}$ and $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, \neg p_i\}$, then $\Sigma \models^* \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_p\}$.*

Definition 4 (U-PRes)
When a clause $C = \{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_p\}$ of Σ satisfies Property 4 w.r.t. the literals p_i to p_j, $U\text{-}PRes(C)$ is defined as $\{\neg n_1, \ldots, \neg n_n, p_1, \ldots, p_{i-1}, p_{j+1}, \ldots, p_p\}$.

Based on the previous properties, a new tractable class called U-Horn SAT is extracted. Algorithm 1 describes how this class can be recognized.

After all Horn clauses have been recorded in Σ', all remaining clauses C are tested successively (line 3). According to Property 3, when the negative part of C is UP-derivable from Σ, this negative part is considered as an additional Horn clause and recorded in Σ' (lines 5 and 6). Else, the second part of Property 3 is implemented in lines 10 to 12. The tests of lines 10 and 15 translate Property 4. In order to obtain $U\text{-}PRes(C)$, the tests in lines 17 to 19 allow the insertion within Σ' of the smallest clause (w.r.t. its number of positive literals). In line 20, a call is made to a procedure described in [13] to get rid of redundant clauses modulo PU. Finally, the initial formula Σ is U-Horn if and only if the simplified formula Σ' is Horn.

Algorithm 1: isU-Horn

Input: a SAT instance Σ
Output: $true$ if Σ is U-Horn; $false$ otherwise

```
1  begin
2    Σ' ← {C|C ∈ Σ s.t. isHorn(C)};
3    forall C ∈ Σ s.t. C = {¬n_1, ..., ¬n_n, p_1, ..., p_p},
4           with n ≥ 0 and p > 1 do
5      if Σ ⊨* {¬n_1, ..., ¬n_n} then
6        Σ' ← Σ' ∪ {{¬n_1, ..., ¬n_n}};
7      else
8        Σ'' ← ∅ ; C' ← C;
9        forall p_i ∈ C do
10         if Σ ⊨* {¬n_1, ..., ¬n_n, p_i} then
11           if Σ ⊨* {¬n_1, ..., ¬n_n, ¬p_i} then
12             Σ' ← Σ' ∪ {{¬n_1, ..., ¬n_n}} ;
13           else
14             Σ'' ← Σ'' ∪ {{¬n_1, ..., ¬n_n, p_i}};
15         else if Σ ⊨* {¬n_1, ..., ¬n_n, ¬p_i} then
16           C' ← C' \ {p_i};
17       if {¬n_1, ..., ¬n_n} ⊄ Σ' then
18         if Σ'' = ∅ then Σ' ← Σ' ∪ {C'};
19         else Σ' ← Σ' ∪ Σ'';
20   Σ' ← redundancyUP(Σ');
21   return isHorn(Σ');
22 end
```

4 Experimental results

In order to assess the practical interest of this pretreatment, a variant of Algorithm 1 (without line 16) has been implemented and experimented. Due to computational reasons, it does not manipulate two CNF (Σ and Σ') as shown in Algorithm 1 but makes use of a same CNF Σ. Consequently, the reduced CNF depends on the order according to which the clauses are considered, and running the program once does not guarantee that all possible simplifications are made, whereas Algorithm 1 ensures this last point. To perform all possible simplifications, the program must be iterated until no new Horn clause is produced. The program has been run on various benchmarks from the DIMACS depository [9] and from the last SAT competitions (`www.satcompetition.org`). All experimentations have been conducted on an Intel(R) Xeon(TM) CPU 3.00GHz with 2Go of memory under Linux CentOS release 4.1. Interestingly enough, some instances were reduced to polynomial-time ones, running the program just once. In Table 1, all instances belonging to the U-Horn class are given: 99 instances belonging to U-Horn class have been found within (almost) 1600 tested instances. For each instance, its name, its size (#var. and #cla.), the number of propagations (#UP) and the time spent in seconds to reduce the instance to U-Horn are given. When the program is iterated until no new Horn clause is produced, 28 additional

CNF instances	# var.	# cla.	#UP	time (s.)	CNF instances	# var.	# cla.	#UP	time (s.)
aim-100-1_6-yes1-4	100	160	179	0	IBM_FV_2004_rule_batch...				
aim-100-2_0-yes1-2	100	200	456	0	IBM_...04_SAT_dat.k15	15300	65598	397812	0.25
aim-100-6_0-yes1-1	100	600	2502	0	IBM_...05_SAT_dat.k15	25128	134922	1708357	1.22
aim-100-6_0-yes1-2	100	600	2534	0	IBM_...15_SAT_dat.k100	226970	893496	2432156	2.46
aim-100-6_0-yes1-3	100	600	777	0	IBM_...15_SAT_dat.k15	30790	119911	184301	0.19
aim-100-6_0-yes1-4	100	600	568	0	IBM_...15_SAT_dat.k20	42330	165416	252596	0.26
aim-200-6_0-yes1-2	200	1200	6113	0.01	IBM_...15_SAT_dat.k25	53870	210921	329216	0.33
aim-200-6_0-yes1-4	200	1200	696	0	IBM_...15_SAT_dat.k30	65410	256426	413391	0.42
aim-50-2_0-yes1-2	50	100	218	0	IBM_...15_SAT_dat.k35	76950	301931	506031	0.5
aim-50-2_0-yes1-3	50	100	250	0	IBM_...15_SAT_dat.k40	88490	347436	606086	0.6
aim-50-2_0-yes1-4	50	100	156	0	IBM_...15_SAT_dat.k45	100030	392941	714746	0.71
aim-50-6_0-yes1-1	50	300	516	0	IBM_...15_SAT_dat.k50	111570	438446	830681	0.83
aim-50-6_0-yes1-2	50	300	692	0	IBM_...15_SAT_dat.k55	123110	483951	955361	0.99
aim-50-6_0-yes1-3	50	300	440	0	IBM_...15_SAT_dat.k60	134650	529456	1087176	1.07
aim-50-6_0-yes1-4	50	300	1621	0	IBM_...15_SAT_dat.k65	146190	574961	1227876	1.22
cnf-r1-b3-k1.2	660004	5281	56944	0.21	IBM_...15_SAT_dat.k70	157730	620466	1375571	1.38
cnf-r1-b4-k1.1	397893	7089	105048	0.18	IBM_...15_SAT_dat.k75	169270	665971	1532291	1.53
cnf-r1-b4-k1.2	922148	6818	60079	0.29	IBM_...15_SAT_dat.k80	180810	711476	1695866	1.69
cnf-r2-b2-k1.2	406052	6064	54402	0.15	IBM_...15_SAT_dat.k85	192350	756981	1868606	1.88
cnf-r2-b3-k1.2	668180	9169	100807	0.27	IBM_...15_SAT_dat.k90	203890	802486	2048061	2.06
cnf-r2-b4-k1.1	406052	12784	178182	0.25	IBM_...15_SAT_dat.k95	215430	847991	2236821	2.26
cnf-r2-b4-k1.2	930282	12464	175575	0.37	IBM_...22_SAT_dat.k10	18919	77414	596987	0.4
jnh10	100	850	6737	0.02	IBM_...22_SAT_dat.k15	29833	122814	1249118	0.96
jnh11	100	850	11187	0.02	IBM_...22_SAT_dat.k20	40753	168249	1845706	1.48
jnh12	100	850	5323	0.01	iso-brn005.shuffled	1130	9866	13572	0.02
jnh13	100	850	4940	0.01	f19-b21-s0-0	746	3517	23805	0.03
jnh14	100	850	3362	0.01	f27-b10-s0-0	193	1113	8268	0.01
jnh15	100	850	7544	0.01	f27-b1-s0-0	193	1113	9401	0.01
jnh18	100	850	16943	0.03	f27-b2-s0-0	193	1113	5614	0.01
jnh19	100	850	10836	0.02	f27-b3-s0-0	193	1113	8716	0.01
jnh202	100	800	4641	0.01	f27-b4-s0-0	193	1113	5992	0.01
jnh203	100	800	18563	0.03	f27-b5-s0-0	193	1113	5626	0.01
jnh208	100	800	16108	0.03	f27-b8-s0-0	193	1113	7702	0.01
jnh20	100	850	8478	0.02	f27-b9-s0-0	193	1113	8684	0.01
jnh211	100	800	3030	0.01	f83-b11-s0-0	1000	43900	318968	0.74
jnh214	100	800	12131	0.02	f83-b14-s0-0	1000	43540	811348	1.61
jnh215	100	800	10558	0.02	f83-b17-s0-0	1000	43900	180456	0.37
jnh216	100	800	12821	0.02	par8-1-c	64	254	5613	0
jnh2	100	850	2201	0	**par8-1**	350	1149	9224	0
jnh302	100	900	246	0	**par8-2**	350	1157	7641	0
jnh303	100	900	13452	0.03	**par8-4-c**	67	266	6216	0
jnh304	100	900	1720	0	**par8-4**	350	1155	10248	0.01
jnh305	100	900	5348	0.01	**par8-5**	350	1171	7978	0
jnh307	100	900	2211	0	pitch.boehm	1192	6361	656	0.01
jnh308	100	900	15155	0.03	qg5-10.shuffled	1000	43900	318968	0.69
jnh309	100	900	2460	0.01	qg6- 10.shuffled	1000	43540	811348	1.62
jnh310	100	900	3054	0.01	qg7-10.shuffled	1000	43900	180456	0.37
jnh4	100	850	5955	0.01	3col20_5_5.shuffled	40	176	774	0
jnh5	100	850	4151	0.01	3col20_5_6.shuffled	40	176	656	0
jnh8	100	850	4749	0.01	3col20_5_7.shuffled	40	176	903	0
jnh9	100	850	3099	0.01	3col20_5_9.shuffled	40	176	438	0

Table 1. U-Horn instances

instances are reduced to U-Horn. They are given in Table 2 where "removed cla" (resp. "removed var") represents the ratio (in percents) of clauses (resp. variables) removed by the method and where "#lit" represents the total number of litterals that have been removed. Even when this pre-treatment does not conduct the instance to be reduced to a polynomial-time one, the global size of the instance is often decreased in a significant manner, whereas its polynomial subpart is increased accordingly. Interestingly, this reduction appears valuable from a global problem-solving point of view. In Table 3, the time required to solve instances using Minisat [12] with the time spent by a combination of the pre-treatment with Minisat are compared. In this table, the columns "Minisat" represent the time consumed by Minisat to solve the original instance ("original") and the simplified one ("simplified"); and the columns "%profit" represents the gain (in percents) obtained by the pre-treatment when the simplification time is taken into account either together with the satisfiability checking time ("total") or not ("partial").

5 Related works

The U-Horn SAT class exhibits a limited similarity with Dalal's Quad fragment [4]. Indeed, both approaches make use of a sub-clauses deduction procedure, using unit propagation inference rules. However, the approach in this paper differs from Dalal's one in several ways. First, it re-

CNF Instances	instance size #var.	#cla.	removed cla	var	#lit.	#UP	time (s.)
een-tipb-sr06-par1	163647	484831	94%	95%	252004	68362283	38.98
ezfact16_10.shuffled	193	1113	26%	34%	335	5614	0.01
ezfact16_3.shuffled	193	1113	37%	44%	479	5992	0.01
f32-b2-s0-0	40	176	70%	69%	178	941	0
f32-b4-s0-0	40	176	85%	77%	163	919	0
f33-b9-s0-0	80	346	88%	80%	391	5867	0
f6-b2-s2-20	478	1007	95%	92%	532	14216	0
IBM_FV_2004_rule_batch_03_SAT_dat.k30	29079	118925	44%	55%	31075	1393665	1.07
IBM_FV_2004_rule_batch_05_SAT_dat.k10	15399	81447	87%	93%	33203	1252239	0.76
IBM_FV_2004_rule_batch_05_SAT_dat.k20	34863	188452	74%	82%	72024	7798669	5.49
IBM_FV_2004_rule_batch_05_SAT_dat.k25	44598	241982	67%	75%	86760	18851484	16.38
IBM_FV_2004_rule_batch_05_SAT_dat.k30	54333	295512	60%	67%	99477	31503131	26.84
IBM_FV_2004_rule_batch_06_SAT_dat.k15	17501	75616	43%	49%	18130	1278040	1.07
IBM_FV_2004_rule_batch_06_SAT_dat.k20	23826	103226	71%	78%	41764	12961178	10.17
IBM_FV_2004_rule_batch_10_SAT_dat.k15	40278	159501	33%	35%	26022	8285670	6.89
IBM_FV_2004_rule_batch_1_11_SAT_dat.k10	28280	111519	47%	49%	25573	58410957	42.46
IBM_FV_2004_rule_batch_18_SAT_dat.k10	17141	69989	48%	55%	19878	13050828	8.7
IBM_FV_2004_rule_batch_19_SAT_dat.k10	21823	83902	24%	31%	13250	298260	0.26
IBM_FV_2004_rule_batch_19_SAT_dat.k15	34697	134023	17%	22%	14917	508638	0.47
IBM_FV_2004_rule_batch_19_SAT_dat.k20	47577	184178	17%	23%	23258	14607263	12.98
IBM_FV_2004_rule_batch_20_SAT_dat.k10	17567	72087	36%	41%	14004	5226452	3.63
IBM_FV_2004_rule_batch_21_SAT_dat.k10	15919	65180	35%	39%	11897	267966	0.21
IBM_FV_2004_rule_batch_21_SAT_dat.k15	25213	103881	25%	28%	13564	471438	0.39
IBM_FV_2004_rule_batch_21_SAT_dat.k20	34513	142616	26%	30%	21454	9624852	7.38
IBM_FV_2004_rule_batch_22_SAT_dat.k25	51673	213684	24%	27%	28739	30219471	22.32
IBM_FV_2004_rule_batch_23_SAT_dat.k10	18612	76086	41%	48%	16035	69713	0.09
IBM_FV_2004_rule_batch_27_SAT_dat.k10	6477	27070	62%	70%	10054	3826810	2.15
rip08.boehm	471	263	92%	59%	145	8728	0.01
x6dn.boehm	521	1255	86%	84%	1022	137818	0.07

Table 2. Reduction of SAT instances using several runs

mains independent from the considered literals ordering. Secondly, a single polynomial fragment is considered instead of several ones in Dalal's work, which as a consequence does not deliver a linear-time pre-treatment. Finally, the use of other treatments based on the removal of redundant clauses [13] and of other reductions operations in our pre-treatment makes the two classes incomparable ones. Obviously enough, the idea of pre-treating SAT instances is not a new one. Many modern SAT solvers include some pre-treatment techniques. For instance, C-SAT [11] made a restricted use of resolution as a polynomial-time pre-treatment, and some DPLL algorithms start with local search runs that, when they fail to prove consistency, are exploited in the further complete search [17]. More recently, Satellite, which is the pre-treatment used in one of the state-of-the-art satisfiability solver, simplifies the instance using variable elimination [1]. Due to its linear-time character, the unit propagation algorithm has been exploited in several ways in the context of SAT, in addition to being a key component of DPLL-like procedures. For example, C-SAT and Satz used a local treatment during important steps of the exploration of the search space, based on UP, to derive implied literals and detect local inconsistencies, and guide the selection of the next variable to be assigned [11, 16]. In [15], a double UP schema is explored in the context of SAT solving. In [19, 14], UP has been used as an efficient tool to detect functional dependencies in SAT instances. The UP technique has also been exploited in [6] in order to derive subclauses by using the UP implication graph of the SAT instance, and speed up the resolution process.

6 Conclusions and perspectives

In this paper, a new linear-time pre-treatment technique for SAT instances has been introduced. It is based on the efficiency of the unit propagation algorithm, which is exploited in order to attempt to increase the polynomial subpart of the targeted SAT instances. Interestingly enough, benchmarks from the SAT competitions that were so far believed to be hard-to-solve problems have been proved to be polynomial SAT instances, and solved accordingly. As such, the pre-treatment is also valuable in that it often increases the efficiency of the satisfiability checking global process. We plan to extend this technique w.r.t. other polynomial fragments of SAT in the future.

Acknowledgments

This research has been supported in part by the CNRS, the EC under a Feder grant and the *Région Nord/Pas-de-Calais*.

References

[1] A. Biere and N. Eén. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the*

	Minisat		% profit		instance size		removed				
CNF Instance	Original	Simplified	partial	total	#var	#cla	var	cla	#lit	#UP	time (s.)
f2clk_40	293.4	265.19	9	7	27568	80439	36%	36%	13619	5009951	5.52
f3-b29-s0-10	76.72	26.58	65	65	2125	12677	24%	35%	3520	127851	0.18
f28-b4-s0-0	3.15	0.04	98	96	769	4777	13%	22%	927	45554	0.08
f81-b3-s0-0	2081.34	1654.61	20	17	33385	163232	26%	30%	24855	36519004	63.69
fifo8_100	14.31	11.12	22	-48	64762	176313	42%	46%	42718	8435460	9.98
fifo8_200	43.74	77.5	-78	-134	129762	353513	37%	40%	76361	18309357	24.51
fifo8_300	349.92	152.11	56	45	194762	530713	35%	39%	109878	28532439	39.94
fifo8_400	500.73	428.59	14	3	259762	707913	34%	38%	143413	38349604	55.85
IBM_03_SAT_dat.k60	28.33	11.99	57	17	59649	244535	22%	27%	33386	12915029	11.33
IBM_03_SAT_dat.k90	195.45	173.44	11	1	90219	370145	17%	20%	38507	21481064	18.32
IBM_05_SAT_dat.k100	204.54	28.02	86	21	190623	1044932	21%	27%	167316	143847825	133.17
IBM_05_SAT_dat.k60	55.59	10.52	81	-37	112743	616692	31%	37%	123076	73459545	65.46
IBM_16_1_SAT_dat.k95	14.62	2.18	85	29	50492	203817	23%	26%	24509	9440470	8.16
ip50	92.63	307.54	-233	-266	66131	214786	36%	44%	47569	23195373	30.61
logistics-rotate-09t6	80.07	6.5	91	-55	8186	887558	15%	30%	908	157186029	117.23

Table 3. Some typical instances

Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05), 2005.

[2] E. Boros, P. L. Hammer, and X. Sun. Recognition of q-horn formulae in linear time. *Discrete Applied Mathematics*, 55(1):1–13, 1994.

[3] M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications-The European Journal for Artificial Intelligence*, 10(3-4):137–150, 1997.

[4] M. Dalal. An almost quadratic class of satisfiability problems. In W. Wahlster, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, pages 355–359, Budapest (Hungary), 1996. John Wiley & Sons, Ltd.

[5] M. Dalal and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44(4):173–180, 1992.

[6] 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.

[7] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

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

[9] Second challenge on satisfiability testing, 1993.

[10] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, pages 267–284, 1984.

[11] O. Dubois, P. André, Y. Boufkhad, and Y. Carlier. *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter SAT vs. UNSAT, pages 415–436. American Mathematical Society, 1996.

[12] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[13] O. Fourdrinoy, É. Grégoire, B. Mazure, and L. Saïs. Eliminating redundant clauses in sat instances. In *Proceedings of the The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, LNCS 4510, pages 71–83, 2007.

[14] 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)*, LNCS 3542, pages 122–132, 2004.

[15] D. Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Fourth International Conference on Theory and Applications of Satisfiability Testing (SAT'01)*, Boston University, Massachusetts, USA, June 2001.

[16] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, 1997.

[17] B. Mazure, L. Saïs, and É. Grégoire. Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, 22:309–322, 1998.

[18] 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.

[19] R. Ostrowski, B. Mazure, L. Saïs, and É. Grégoire. Eliminating redundancies in SAT search trees. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2003)*, pages 100–104, Sacramento, November 2003.

[20] B. Selman and H. A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[21] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.

Local-Search Extraction of MUSes

Article publié dans la revue « *Constraints* » volume 12, n°3, pages 325-344, 2007.

Co-écrit avec Éric GRÉGOIRE et Cédric PIETTE.

Local-Search Extraction of MUSes

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

Abstract

SAT is probably one of the most-studied constraint satisfaction problems. In this paper, a new hybrid technique based on local search is introduced in order to approximate and extract minimally unsatisfiable subformulas (in short, MUSes) of unsatisfiable SAT instances. It is based on an original counting heuristic grafted to a local search algorithm, which explores the neighborhood of the current interpretation in an original manner, making use of a critical clause concept. Intuitively, a critical clause is a falsified clause that becomes $true$ thanks to a local search flip only when some other clauses become $false$ at the same time. In the paper, the critical clause concept is investigated. It is shown to be the cornerstone of the efficiency of our approach, which outperforms competing ones to compute MUSes, inconsistent covers and sets of MUSes, most of the time.

1 Introduction

SAT is probably one of the most-studied constraint satisfaction problems: it consists in checking whether a set of Boolean clauses admits at least one truth assignment that satisfies all clauses. There is a large and very active CSP-related research community involved with both the theoretical and experimental algorithmic studies of SAT, and its application to various computer-science domains (see e.g. `http://www.SATlive.org`).

In this paper, a new hybrid technique based on local search (in short, LS) is introduced to approximate and extract minimally unsatisfiable subformulas (in short, MUSes) of unsatisfiable SAT instances. Extracting MUSes can prove valuable in many applications, because it circumscribes what is *wrong* with an instance. For example, when we check the logical consistency of knowledge-bases, we prefer knowing which pieces of information are contradicting one another rather than only knowing that the whole base is contradictory. MUSes provide such a kind of information, as they represent the smallest explanations -in terms of the number of involved constraints- of infeasibility. Indeed, many application domains like model-based diagnosis, knowledge-base validation or VLSI correctness checking, require such explanations to be delivered in case of infeasibility.

Interesting enough, our technique is based on an original counting heuristic grafted to a local search algorithm, which explores the neighborhood of the current interpretation in an original manner, making use of the critical clause concept. Intuitively, a critical clause is a falsified clause that becomes $true$ thanks to a local search flip only when some other clauses become $false$ at the same time. In the paper, this concept is investigated. It is shown to be the key to the efficiency of our approach, which outperforms competing ones to approximate and compute MUSes most of the time. Although it is based on a heuristic and local search, our technique is complete in the sense that it always delivers a MUS for any unsatisfiable SAT instance.

In the second part of the paper, the technique is extended to the issue of extracting strict inconsistent covers, which consists of a set-theoretical union of independent MUSes that would allow the SAT instance to become satisfiable if this set were removed from the instance. Such a concept is introduced as a viable trade-off between a number of MUSes that can be exponential in the size of the instance, and the need to provide a computational approach that delivers enough infeasibility causes in order for the unsatisfiability to be fully explained.

The paper is organized as follows. In the next section, the necessary technical background is provided, and the MUS and inconsistent cover concepts are presented formally. Then, the complexity properties of computing MUSes are recalled, and our study is positioned with respect to the major related works. In section 4, the crucial notion of critical clause is introduced and analyzed. In section 5, the new approach to approximate or compute one MUS is presented. Extensive experimental results are given in section 6. Next, the approach is extended to compute strict inconsistent covers. Before we conclude, section 8 shows how the approach can be extended to compute sets of MUSes, or even compute all MUSes of a SAT instance.

2 MUSes and Inconsistent Covers

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables, denoted a, b, etc. Formulas will be denoted using upper-case letters such as C. Sets of formulas will be represented using Greek letters like Γ or Σ. An interpretation is a truth assignment function that assigns values from $\{true, false\}$ to every Boolean variable. A formula is consistent or satisfiable when there is at least one interpretation that satisfies it, i.e. that makes it become $true$. An interpretation will be denoted by upper-case letters like I and will be represented by the set of literals that it satisfies. Actually, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) using a set (interpreted as a conjunction) of clauses, where a clause is a finite disjunction of literals and where a literal is Boolean variable that can be negated. SAT is the canonical NP-complete problem that consists in checking whether a set of Boolean clauses is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not.

Let us also recall the SAT-related optimization problem, namely max-SAT.

Definition 1 *Given a SAT instance Σ, max-SAT consists in finding a maximal number of clauses of Σ that can be satisfied under a same interpretation.*

When a SAT instance is unsatisfiable, it exhibits at least one minimally unsatisfiable subformula, in short one *MUS*.

Definition 2 *A MUS Γ of a SAT instance Σ is a set of clauses s.t.*

1. $\Gamma \subseteq \Sigma$
2. Γ *is unsatisfiable*
3. *Every proper subset of* Γ *is satisfiable*

In the following, another crucial concept is the notion of *(strict) inconsistent cover*.

Definition 3 *Two sets of clauses are independent iff their intersection is empty. One (strict) inconsistent cover IC of an unsatisfiable SAT instance Σ is a set-theoretic union of (independent) MUSes of Σ s.t. $\Sigma \backslash IC$ is satisfiable.*

It should be noted that a same unsatisfiable instance can exhibit several different strict inconsistent covers. For example, let $\Sigma = \{a, b, \neg a \vee \neg b, \neg a, c, \neg c \vee a\}$. Σ contains 3 MUSes: $MUS_1 = \{a, \neg a\}$, $MUS_2 = \{a, b, \neg b \vee \neg a\}$ and $MUS_3 = \{\neg a, c, \neg c \vee a\}$. Σ contains 2 strict inconsistent covers, namely $IC_1 = MUS_1$ and $IC_2 = MUS_2 \cup MUS_3$.

Although a strict inconsistent cover does not provide us with the set of all MUSes that may be present in a formula, it encodes a series of minimal explanations for unsatisfiability that are sufficient to explain enough sources of infeasibility in order for the whole formula to regain satisfiability if they were all fixed.

A straightforward result is that a strict inconsistent cover enables us to get a lower-bound of the number of clauses that are assigned $false$ in a max-SAT solution of an unsatisfiable SAT instance.

Property 1 *Let Σ be an unsatisfiable SAT instance. Let IC be a strict inconsistent cover of Σ. Let $|IC|$ be the number of independent MUSes contained in IC. For any interpretation I of Σ, at least $|IC|$ clauses of Σ are $false$ under I.*

Proof *By definition, strict inconsistent covers contain MUSes with empty intersection. Since at least one clause per MUS is assigned $false$ under any interpretation I, at least $|IC|$ clauses are thus falsified under I.* □

3 Computational Complexity and Related Works

The worst-case complexity of computing MUSes is high. Indeed, checking whether a set of clauses is a MUS or not is DP-complete [26], and checking whether a formula belongs to the set of MUSes of an unsatisfiable instance or not, is in Σ_2^p [10]. Moreover, the number of MUSes can be exponential in the size of the instance. Indeed, the cardinality of the set of MUSes of an n-clauses instance is $C_n^{n/2}$ in the worst case. However, let us stress that the number of MUSes remains often tractable in real-life situations. For example, in model-based diagnosis [15], based on experimental studies, it is often assumed that single faults occur, which are often expressed through a small number of MUSes.

Recently, several approaches have been proposed to approximate or compute MUSes. Unfortunately, they concern specific classes of clauses or they remain tractable for small instances only. Among them, Bruni [6] has shown how a MUS can be extracted in polynomial time through linear programming techniques for clauses exhibiting a so-called integral point property. However, only restrictive classes of clauses obey such a property (mainly Horn, renamable Horn, extended Horn, balanced and matched ones). Other studies about the computational complexity and the algorithmic properties of extracting MUSes for specific classes of clauses can be found in [2, 8, 11]. In [5], Bruni has also proposed an approach that approximates MUSes by means of an adaptative search guided by clauses hardness. Zhang and Malik have described in [28] a way to extract MUSes by learning nogoods involved in the derivation of the empty clause by resolution. In [21], Lynce and Marques-Silva have proposed a complete and exhaustive technique to extract one smallest MUS of a SAT instance. Together with Mneimneh, Andraus and Sakallah [24], the same authors have also proposed an algorithm that makes use of iterative max-SAT solutions to compute such smallest unsatisfiable subsets of clauses. Oh and her co-authors have presented in [25] a Davis, Putnam, Logemann and Loveland DPLL-oriented approach that is based on a marked clause concept to allow one to approximate MUSes. Liffiton and Sakallah have shown how MUSes can be computed through the dual concept of maximally satisfiable subsets [20]. In [12], this latter approach has been successfully hybridized with an appropriate local search technique, in order to get some strategic information allowing both NP et CoNP tests to be avoided, reducing the global computing time most often.

In [23], a heuristic was also proposed to approximate MUSes, based on the experimental finding that clauses that are most often falsified during a failed local search often belong to MUSes. It has also been used to improve the performance of DPLL-like complete algorithms [7]. To some extent, the study presented in this paper can be understood as a powerful variant and extension of this former technique.

4 A New Heuristic to Detect MUSes

In [23], it was first shown how local search can be helpful for approximating MUSes. The basic idea is that clauses that are often falsified during a failed local search for satisfiability belong most probably to MUSes, when the instance is actually unsatisfiable. When the score of a clause is the number of times it has been falsified during a failed local search (in short, failed LS), discriminating the clauses with a high score can deliver a good approximation of the set of MUSes. Such a heuristic has been studied in an extensive manner in [22] and [23]. It has also been extended in several ways to address decision and optimization problems that belong to higher levels of the polynomial hierarchy (see e.g. [13, 4, 14, 3]).

In the following, the SAT instance is assumed to be unsatisfiable. The aforementioned heuristic can require us to increment the score of clauses even when they do not actually belong to any MUS. Unless we solve the problem of finding MUSes itself, we can only rely on some heuristic indications about the extent to which a currently falsified clause could or could not belong to a MUS. In this respect, we claim that some relevant parts of the neighborhood of the current interpretation can be checked and pro-

vide more information about whether a currently falsified clause C should be counted or not. The idea is to take the structure of C into account and to increment the score of C only when it cannot be satisfied without conducting other clauses to be falsified in their turn. We shall see that this technique implements definitions that approximate a property that is intrinsic to clauses belonging to MUSes.

To illustrate this concept, let us consider the following example. Let $\Delta = \{a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c\}$. Δ is unsatisfiable and is its own MUS. Let $I = \{a, b, c\}$ be an interpretation. Under this interpretation, only the $\neg a \vee \neg b \vee \neg c$ clause is falsified. In the following, the *once-satisfied* clause concept will prove useful.

Definition 4 *A clause C is once-satisfied under an interpretation I iff exactly only one literal of C is satisfied under I.*

In the above example, the clauses $\neg a \vee b$, $\neg b \vee c$ and $\neg c \vee a$ are once-satisfied under $I = \{a, b, c\}$.

Definition 5 *A clause C falsified under the interpretation I is critical w.r.t. I iff the opposite of every literal of C belongs to a clause that is once-satisfied under I. These once-satisfied clauses that are not tautological ones are called linked to C.*

In the example, $\neg a \vee \neg b \vee \neg c$ is both falsified under I and critical w.r.t. I; its related linked clauses are the once-satisfied ones $\neg a \vee b$, $\neg b \vee c$ and $\neg c \vee a$.

The role of these definitions is easily understood thanks to the following property.

Property 2 *Let C be a critical clause w.r.t. an interpretation I, then any flip from I to I' such that C is satisfied under I' will conduct I' to falsify at least one clause that was satisfied under I.*

Proof *If C is critical then for each literal l of C, $\exists\ C'$ s.t. C' is once-satisfied w.r.t. I and $\bar{l}$ belongs to C'. C is falsified under I, thus l is $false$ under I and $\bar{l}$ is $true$ under I. $\bar{l}$ is the only satisfied literal of C', accordingly if the value of l is reversed then C' becomes falsified.* □

In order to discriminate clauses belonging to MUSes, the idea is to increment the scores of critical clauses during the search, together with their linked (satisfied) clauses, rather than increment the scores of all falsified clauses. Such a technique can be easily grafted to a LS algorithm and the updates can be easily computed. Actually, it implements definitions that approximate a property that is obeyed by clauses belonging to MUSes.

Property 3 *Let I be an interpretation giving an optimal result for max-SAT on an unsatisfiable instance Σ. Then, any falsified clause C under I belongs to at least one MUS of Σ and is critical w.r.t. I. Moreover, at least one once-satisfied clause linked to C also belongs to a MUS of Σ.*

Proof *Any falsified clause under I belongs to a MUS because I is optimal w.r.t. the number of satisfied clauses and at least one clause of each MUS cannot be satisfied.*

The fact that any falsified clause under I is critical is proved thanks to Property 5 since I is a global mimimum. I is optimal with respect to the number of satisfied clauses, thus at most one clause per MUS is falsified. Also, if one flip allows us to satisfy one of theses clauses, another clause of the MUS becomes falsified. Accordingly, at least one once-satisfied clause linked to a falsified clause under I belongs to a MUS of Σ. □

Counting critical clauses during a local search run can thus be interpreted as trying to find an approximation of a MUS in the sense that clauses and their linked ones are considered during the whole search, and not w.r.t. an interpretation delivering a solution of max-SAT. Indeed, being a critical clause is neither a necessary nor a sufficient condition to belong to a MUS. As the following example illustrates, a critical clause w.r.t. an interpretation that is not an optimal one w.r.t. max-SAT for an unsatisfiable formula might not belong to a MUS. Let $\Delta = \{a \vee d, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$. Clearly, Δ is satisfiable. $\neg e \vee \neg f$ is falsified under $I = \{a, b, d, e, f\}$ and is critical w.r.t. I. Moreover, a clause from a MUS that is falsified under a given interpretation I is not necessary critical w.r.t. I, as the following example shows. Let $\Delta = \{a \vee d, b, \neg a \vee \neg b, \neg d \vee e, f, \neg e \vee \neg f\}$. Clearly, Δ is a minimal unsatisfiable set of clauses. $\neg a \vee \neg b$ is falsified under $I = \{a, b, d, e, f\}$. However, it is not critical w.r.t. I. Fortunately, the following property ensures that all clauses from a MUS can be scored by the heuristic.

Property 4 *Let Γ be a MUS. For all clauses $C \in \Gamma$, there exists an interpretation I s.t. C is critical w.r.t. I.*

Proof *Let Γ be a MUS and C be a clause s.t. $C \in \Gamma$. By definition of a MUS, $\Gamma \setminus C$ is satisfiable. Let M be a model of $\Gamma \setminus C$. Let us prove that C is critical w.r.t. M. First, C is falsified. Indeed, if C is not falsified then Γ exhibits a model M. This is impossible because Γ is a MUS. Second, C is critical. Indeed, if any variable occurring in C is flipped w.r.t. M, then at least one clause of Γ becomes unsatisfied since Γ is unsatisfiable. That means that this new unsatisfied clause was once-satisfied and linked to C. Accordingly, C is critical w.r.t. M.* □

This property ensures that any clause that takes part to a MUS can be critical w.r.t. at least one interpretation. As such, this property does not guarantee that our scoring heuristic will allow us to exhibit all clauses belonging to MUSes. Indeed, it does not indicate that a LS run will necessary increment the score of all such clauses at least once since LS does not necessary visit all interpretations. However, the following property and its corollary provide us with a good indication that LS will probably visit interpretations where clauses belonging to MUSes are critical. Indeed, it is well-known that LS is in general attracted by local minima. Property 5 ensures that all falsified clauses are critical in local or global minima.

Definition 6 *A local minimum is an interpretation s.t. no flip can increase the number of satisfied clauses. A global minimum (or max-SAT solution) is an interpretation delivering a maximal number of satisfied clauses.*

Property 5 *In (local or global) minima, all falsified clauses are critical.*

Proof *If a variable occurring in a falsified clause w.r.t. a minimum is flipped, then this clause is satisfied and at least one previously satisfied clause becomes falsified, since in minima no flip can increase the number of satisfied clauses. That means that this new unsatisfied clause was once-satisfied. Accordingly, the initial falsified clause was critical.* □

A corollary ensures that at least one clause per MUS is critical in such minima.

Corollary 1 *In (local or global) minima, at least one clause per MUS is critical.*

Let us illustrate the possible role of critical clauses in computing MUSes through an example.

Example 1 *Let Σ be a SAT instance such that :*

$$\Sigma = \begin{cases} \underline{a} \vee \underline{b} \vee c, & \leftarrow \\ \neg b \vee e, & \\ \neg a \vee \underline{b} \vee c, & \leftarrow \\ \neg a \vee \neg b, & \leftarrow \\ \underline{a} \vee d, & \\ \underline{b} \vee \underline{\neg c}, & \leftarrow \\ \underline{\neg d} \vee e, & \\ \underline{a} \vee \neg b, & \leftarrow \\ \underline{\neg e} \vee \underline{\neg f} & \end{cases}$$

Σ is unsatisfiable and has a unique MUS. Clauses that form this MUS are postfixed by an arrow. Let $I = \{\neg a, \neg b, c, d, e, f\}$ be the current interpretation explored by a local search run. $b \vee \neg c$ and $\neg e \vee \neg f$ are the only clauses of Σ that are falsified under I. Each falsified litteral is underlined. If a counting heuristic considers all falsified clauses in order to approximate and compute a MUS, then these two clauses will have their counters incremented although only $b \vee \neg c$ belongs to the MUS. Considering the neighborhood of the current interpretation using the critical clause concept enables us to score the right clause only. In fact, $b \vee \neg c$ is critical w.r.t. I whereas the other falsified clause is not. Indeed, if we turn it to $true$ by flipping c in I, then $a \vee b \vee c$ becomes falsified, as it was once-satisfied w.r.t I. A similar phenomenon happens if we flip b. Let us stress that these two once-satisfied linked clauses also belong to the unique MUS of Σ. On the contrary, $\neg e \vee \neg f$ can be satisfied by performing one flip without falsifying any another clause. It should thus not be scored in order to augment its probable MUS-membership status.

5 Approximating and Computing one MUS

In the following, it is shown that a meta-heuristic based on scoring critical clauses is viable in order to approximate or compute MUSes. Actually, due to implementation

Algorithm 1: AOMUS algorithm

Input: an unsatisfiable SAT formula Σ
Output: an Approximation of One MUS of Σ

```
1  begin
2      stack ⟵ ∅ ;
3      while (LS + Score(Σ) fails to find a model) do
4          push(Σ) ;
5          Σ ⟵ Σ \ LowestScore(Σ) ;
6      repeat
7          Σ ⟵ pop() ;
8      until Σ is UNSAT ;
9      return Σ ;
10 end
```

Algorithm 2: `fine-tune` procedure

Input: an approximation of a MUS of Σ
Output: a MUS extracted from Σ

```
1  begin
2      foreach clause c ∈ Σ sorted according to their scores do
3          if Σ \ {c} is unsatisfiable then
4              Σ ⟵ Σ \ {c} ;
5      return Σ ;
6  end
```

efficiency constraints, we update the scores of critical clauses only. Updating the scores of their linked clauses does not lead to dramatic performance improvements, at least w.r.t. our selected LS algorithm and tested benchmarks.

The main idea is as follows. Let Σ be an unsatisfiable SAT instance. While local search fails to find a model of Σ, we remove clauses of Σ that have the lowest scores. We record the obtained sub-formulas on a stack. Next, the unsatisfiability of the last subformula where LS fails to find a model is checked. If this subformula is unsatisfiable, then it is an approximation of a MUS of Σ. Otherwise, this unsatisfiability check is repeated on the lastly recorded supersets of clauses, successively, until one such set is proven unsatisfiable. This algorithm is called AOMUS (Approximate One MUS) and is described in Algorithm 1.

Then an exact MUS can be obtained by a step-by-step minimization of the upper-approximation until the remaining clauses are proven to form a MUS (see [16, 17] for alternative methods that do not exploit information delivered by a previous LS). This process is called `fine-tune` and is described in Algorithm 2. The order according to which clauses are tested can be guided by the score of each clause.

Let us stress that this approach is complete in the sens that it will always deliver one MUS for any unsatisfiable instance. The combination of AOMUS with the `fine-tune`

Algorithm 3: OMUS algorithm

Input: an unsatisfiable SAT formula Σ
Output: One MUS of Σ

```
1 begin
2 |   Σ ⟵ AOMUS(Σ) ;
3 |   Σ ⟵ fine-tune(Σ) ;
4 |   return Σ ;
5 end
```

procedure is called OMUS (find One MUS) and is described in Algorithm 3. Its efficiency directly depends on the quality of the upper-approximation. In the next section, experimental results show that the approximation delivered by AOMUS is often of a good quality, because a very small set of clauses is removed by the fine-tune step and in consequence a very small number of unsatisfiability tests are performed (when a clause belongs to the MUS, the test amounts to a satisfiability check).

Actually, we refined this basic procedure in the following manner. Assume that the current computed subformula is actually unsatisfiable; whenever a unique clause remains falsified during the LS run, we are sure that this clause belongs to all MUSes of this current subformula.

We mark these clauses as *protected* and they cannot be removed from Σ thereafter. Moreover, this information is kept all along the process because it can prove very useful during the minimization procedure. After the approximation is computed, the idea is to remove each clause to check whether it partipates to the infeasibility cause of the formula or not. However, protected clauses do not need to be tested, because we know that removing one of them restores satisfiability. Furthermore, when the remaining falsified clauses contain protected clauses only, they form one exact MUS. In this case, the fine-tune step can be omitted since we already have extracted an exact MUS of the formula.

In [19], the clauses of an unsatisfiable CNF formula are classified according to their role in inconsistency. Specifically, clauses that belong to every resolution refutation are called *necessary*, and removing any of those clauses from the formula enables to regain satisfiability. To some extent, this concept of necessary clause can be related to the protected one, since this latter one consits in marking, in an incomplete way, clauses such that all the constraints can be satisfied except them. Hence, removing any of them enables to regain satisfiability of the current generated subformula: protected clauses are thus necessary clauses of this subformula and belong to any of its MUSes, and to any MUS of its unsatisfiable subsets.

It appears that this refinement proves useful for many instances' and allows a dramatic efficiency gain for both the AOMUS and OMUS procedures.

The parameters that were selected for these methods are as follows. As a case study, Walksat [18] with the $Rnovelty+$ option was chosen as the LS procedure. The following parameters were fine-tuned based on extensive tests on various benchmarks. After each flip of the LS, the score of critical clauses is increased by the number of their linked clauses. This technique allows us to take the length of critical clauses

into account, since the number of linked clauses depends on the length of the critical clause in terms of the number of involved literals. Now, clauses whose score is lower than $(min\text{-}score + \frac{\#Flips}{\#Clauses})$ are dropped, where $min\text{-}score$ is the lowest score for a clause of Σ; $\#Flips$ and $\#Clauses$ are the number of performed flips and the number of clauses in Σ, respectively.

This procedure was tested extensively on various unsatisfiable instances from several difficult benchmarks from DIMACS [9] and from the annual SAT competitions [27], and compared with other published approaches to compute MUSes, as described in the next section.

6 Experimental Results

Our algorithms have been implemented and compared with previously proposed approaches on numerous benchmarks. All experiments have been conducted on Pentium IV, 3Ghz with 1 Go of memory under linux Fedora Core 4. Time-out indicates that no result has been obtained within 1 hour CPU time.

First, AOMUS has been compared to the the currently best algorithms that extract an unsatisfiable subformula that is not guaranteed to be a minimal one. Second, the focus has been on methods that extract exact MUSes. Except for Bruni's results which are just size results that we have extracted from [5], the experimental size of the discovered smallest unsatisfiability subsets are provided, together with the CPU time in seconds to get them.

Thus, the first part of our experimental study concerns approximate methods, namely procedures that provide unsatisfiable subformulas without ensuring that they are minimal ones. First, AOMUS has been compared to a possible variant, where $Score$ is the basic heuristic from [23], which simply counts the number of times a clause is falsified. Not surprisingly, our approach proves more efficient. We have also compared our approach with `zCore`, the core extractor of `zChaff` [28]. `zChaff` is currently one of the most efficient SAT solvers. We have also ran Lynce and Marques-Silva's procedure [21], the AMUSE algorithm [25], and have taken Bruni's [5] experimental results into account. For Bruni's technique, we only mention the experimental results obtained by the author, since this system is not available. Although a comparison with Bruni's technique is thus difficult to achieve from an experimental side, it appears that Bruni's technique has been experimented on small instances, only. A typical sample of experimental results are reported in Table 1, which illustrates how AOMUS proves more competitive with respect to both the required run-times and the quality of the delivered approximations of MUSes, in the sense that smaller-sizes approximations are obtained most often.

`zCore` proves quite efficient for single-MUS instances but fails to deliver good results when several MUSes are present. Indeed, `zCore` does not concentrate on finding one MUS, but on finding proofs of unsatisfiability. AMUSE also behaves very well on some benchmarks but proves often less competitive when very large and difficult multi-MUSes instances are considered, both with respect to run-time results and the quality of the obtained approximation. For example, let us consider the FPGA switch-box series of problems. The `fpga_11_*` instances have various sizes, in terms of

Table 1: Approximate One MUS (AOMUS) vs. previous approaches

Instance	#var	#cla	AMUSE [25]		Bruni [6]	zCore [28]		Scoring like [23]		AOMUS	
			#cla	Time	#cla	#cla	Time	#cla	Time	#cla	Time
fpga10_11	220	1122	561	73.93	-	561	28.51	561	18.26	561	13.06
fpga10_12	240	1344	663	104.28	-	672	71.27	561	30.11	561	16.9
fpga10_13	260	1586	561	74.61	-	793	166.99	561	51.67	561	25.95
fpga10_15	300	2130	561	80.61	-	1065	570.3	561	128.05	561	44.18
fpga11_12	264	1476	738	928.40	-	738	112.53	738	66.8	738	65.49
fpga11_13	286	1742	870	1971.77	-	871	504.97	738	180.66	738	56.71
fpga11_14	308	2030	856	1200.37	-	1015	1565.6	738	415.32	738	69.55
fpga11_15	330	2340	738	1008.08	-	Time out		738	568.79	738	52.14
aim100-1_6-no-2	100	160	54	0.07	54	54	0.05	53	0.27	53	0.38
aim100-2_0-no-1	100	200	19	0.05	19	19	0.09	19	0.22	19	0.19
aim200-1_6-no-3	200	320	83	0.06	86	83	0.07	83	0.37	83	0.44
aim200-2_0-no-3	200	400	37	0.08	37	37	0.23	37	0.39	37	0.49
aim50-1_6-no-4	50	80	20	0.06	20	20	0.04	20	0.16	20	0.16
aim50-2_0-no-4	50	100	21	0.07	21	21	0.14	21	0.21	21	0.22
2bitadd_10	590	1422	929	148.27	-	815	343.48	1212	42.752	806	189.47
barrel2	50	159	79	0.22	-	77	0.04	100	0.35	77	0.36
jnh10	100	850	119	0.09	161	68	0.88	128	9.35	79	42.25
jnh20	100	850	141	0.11	120	102	0.23	104	21.68	87	48.93
jnh5	100	850	151	0.09	125	86	0.39	140	12.653	88	46.2
jnh8	100	850	156	0.09	91	90	0.22	162	28.964	69	90.53
SGI_30_80_16_90_1	480	30464	Time out		-	Time out		Time out		Time out	
homer06	180	830	415	34.96	-	415	15.96	415	10.97	415	9.04
homer07	198	1012	506	37.66	-	506	21.6	415	12.59	415	10.67
homer08	216	1212	506	52.42	-	606	44.46	554	23.43	415	19.79
homer09	270	1920	832	217.15	-	960	141.48	415	93.19	504	60.9
homer10	360	3460	1096	195.87	-	940	624.11	1614	148.27	503	466.94
homer11	220	1122	561	324.13	-	561	23.44	561	41.68	561	15.6
homer12	240	1344	672	575.54	-	672	76.19	708	25.92	564	41.03
homer13	260	1586	793	930.83	-	793	152.13	579	67.38	561	76.66
homer14	300	2130	793	999.18	-	1065	714.03	561	347.19	561	28.03
homer15	400	3840	1546	3426.76	-	Time out		677	247.84	561	1048.28
homer16	264	1476	Time out		-	738	115.49	738	78.44	738	61.31
homer17	286	1742	Time out		-	871	369.11	870	127.43	738	68.28
linvrinv2	24	61	51	0.07	-	51	0.05	57	0.17	53	0.11
linvrinv3	90	262	253	0.09	-	250	0.15	250	0.41	244	0.32
linvrinv4	224	689	689	4.15	-	689	22.19	689	21.35	657	19.11
marg5x5	105	512	Time out		-	Time out		Time out		Time out	

More extensive results are available at `http://www.cril.fr/~piette`

both the numbers of involved variables and clauses, but all of them contain the same 738-clauses MUS. AMUSE and zCore get into trouble when they need to focus on this MUS when the selected instances are large. Actually, the returned subformula is just a superset of this MUS and the run-times increase dramatically with the size of the instance. On the contrary, AOMUS always delivers this MUS within a reasonable time.

The second part of our experiments has been devoted to exact methods. To the best of our knowledge, only one algorithm has been previously proposed in order to compute one exact MUS. It is called zminimal and is based on the zCore procedure. Actually, it computes an upper-approximation of a MUS thanks to a zCore call, and then reduces it using step-by-step tests, like our finetune procedure. Both approaches have been compared and typical experimental results are given in Table 2.

Once again, our approach proves very competitive compared to zminimal, since it was able to deliver a MUS within a reasonable time for all considered families of in-

Table 2: Compute One MUS (OMUS) vs. zminimal

Instance	#var	#cla	zminimal		OMUS	
			#cla	Time	#cla	Time
fpga10_11	220	1122	561	49.83	561	13.75
fpga10_12	240	1344	643	548.77	561	17.03
fpga10_13	260	1586	643	3406.71	561	31.89
fpga10_15	300	2130	Time out		561	68.17
fpga11_12	264	1476	738	105.61	738	66.3
fpga11_13	286	1742	Time out		738	84.74
fpga11_14	308	2030	Time out		738	304.4
fpga11_15	330	2340	Time out		738	85.2
aim100-1_6-no-2	100	160	53	0.10	53	0.38
aim100-2_0-no-1	100	200	19	0.06	19	0.23
aim200-1_6-no-3	200	320	83	0.23	83	0.83
aim200-2_0-no-3	200	400	46	0.09	37	0.54
aim50-1_6-no-4	50	80	20	0.05	20	0.17
aim50-2_0-no-4	50	100	28	0.06	21	0.27
2bitadd_10	590	1422	Time out		716	268.5
barrel2	50	159	77	0.09	77	0.44
jnh10	100	850	67	0.31	79	42.9
jnh20	100	850	99	0.21	87	75.76
jnh5	100	850	86	0.17	86	46.87
jnh8	100	850	85	0.16	67	99.07
SGI_30_80_16_90_1	480	30464	Time out		Time out	
homer06	180	830	415	15.64	415	9.3
homer07	198	1012	415	540.26	415	19.19
homer08	216	1212	552	572.69	415	24.65
homer09	270	1920	Time out		415	81.23
homer10	360	3460	Time out		415	513.11
homer11	220	1122	561	23.94	561	16.32
homer12	240	1344	561	1426.21	561	62.34
homer13	260	1586	Time out		561	78.51
homer14	300	2130	Time out		561	30.64
homer15	400	3840	Time out		561	1104.13
homer16	264	1476	738	328.20	738	62.91
homer17	286	1742	Time out		738	87.4
linvrinv2	24	61	51	0.07	51	0.17
linvrinv3	90	262	240	0.83	240	0.63
linvrinv4	224	689	651	666.98	651	133.21
marg5x5	105	512	Time out		Time out	

More extensive results are available at http://www.cril.fr/∼piette

stances, whereas zminimal could not provide such a result within 1 hour CPU time for the largest and hardest benchmarks. It should be emphasized that the step-by-step procedures of both zminimal and OMUS are similar ones. Thus, the experimental results can be explained as follows. First, AOMUS delivers approximations that are often of a really better quality than the approximations obtained using zCore. It follows that the OMUS minimization step requires a fewer number of calls to NP and CoNP procedures. Moreover, the approximations delivered by AOMUS are often built using a smaller set of variables, making the subsequent calls to complete methods more efficient. Second, as explained above, AOMUS also delivers a set of so-called 'protected' clauses, which do not need to be further tested since it is already known that they belong to the MUS. Accordingly, a lot of tests can be avoided. In fact, on some problems, all clauses were found protected and no further test was needed to conclude that one exact MUS has already been extracted using AOMUS.

Let us stress that the MUSes discovered by these two approaches are not necessary the same ones (see e.g. the homer series). Moreover, OMUS extracts smaller MUSes

Algorithm 4: `ICMUS` algorithm

Input: an unsatisfiable SAT formula Σ
Output: a strict Inconsistent Cover of Σ

```
1 begin
2     IC ⟵ ∅ ;
3     while Σ is unsatisfiable do
4         MUS ⟵ OMUS(Σ) ;
5         IC ⟵ IC ∪ MUS ;
6         Σ ⟵ Σ \ MUS ;
7     return IC ;
8 end
```

than `zminimal` does, most often.

7 Extracting Strict Inconsistent Covers

These last years, several approaches have also been proposed to compute *all* MUSes of a SAT instance (e.g. [1, 20]). Unfortunately, these computational approaches remain often intractable since, among other things, the number of MUSes in a formula can be exponential in the size of the formula. In this section, a novel method is introduced to compute *independent* MUSes, i.e. MUSes that do not share any clause. The idea motivating this approach is that independent MUSes express independent −uncorrelated− causes of unsatisfiability inside a same formula. This lead us to the concept of *strict inconsistent cover* of an unsatisfiable instance.

Although an inconsistent cover does not provide us with the set of all MUSes that may be present in a formula, it gives us a series of minimal explanations for infeasibility that are sufficient to explain and potentially repair enough sources of unsatisfiability in order for the whole formula to regain satisfiability. Moreover, formulas can have MUSes that are very small with respect to the size of the formula; dropping or repairing such MUSes can sometimes be sufficient to regain satisfiability. Since strict inconsistent covers are composed of independent MUSes, one way to obtain a strict inconsistent cover is to compute a single MUS, then remove it from the formula, and repeat these operations until the remaining subformula becomes satisfiable. This method is called `ICMUS` (find a strict Inconsistent Cover of MUSes) and described in Algorithm 4.

In Table 3, some typical experimental results are provided. Unsurprisingly, a lot of instances exhibit very small inconsistent covers in terms of the number of involved clauses. For example, let us consider the `ezfact16_2` instance. This formula contains 1113 clauses, and possesses one MUS that is composed of 41 clauses. This MUS is in fact a strict inconsistent cover because its removal restores the satisfiability of the formula. The industrial-related formula `term1_gr_rcs_w3` exhibits a strict inconsistent cover made of 11 small MUSes. Thanks to this result, we can deduce that all interpretations falsify at least 11 clauses. Indeed, as shown in Property 1 a strict incon-

Table 3: Inconsistent covers for different classes of formulas

Instance	#var	#cla	Time	**#MUSes** in the IC	(#var,#cla) for each MUS
dp02u01	213	376	1.19	**1**	(47,51)
dp03u02	478	1007	362	**1**	(327,760)
23.cnf	198	474	2.68	**1**	(165,221)
42.cnf	378	904	9	**1**	(315,421)
fpga10_11_uns_rcr	220	1122	56	**2**	(110,561) (110,561)
fpga11_12_uns_rcr	264	1476	128	**2**	(132,738) (132,738)
ca002	26	70	0.61	**1**	(20,39)
ca004	60	168	1.11	**1**	(49,108)
ca008	130	370	5.26	**1**	(110,255)
term1_gr_rcs_w3	606	2518	6180	**11**	(12,22) (21,33) (30,58) (12,22) (12,22) (12,22) (12,22) (12,22) (12,22) (24,39) (21,33)
C220_FV_RZ_14	1728	4508	28	**1**	(10,14)
C220_FV_RZ_13	1728	4508	46	**1**	(9,13)
C170_FR_SZ_96	1659	4955	18	**1**	(81,233)
C208_FA_SZ_121	1608	5278	21	**1**	(18,32)
C168_FW_UT_851	1909	7491	83	**1**	(7,9)
C202_FW_UT_2814	2038	11352	304	**1**	(15,18)
jnh208	100	800	14	**1**	(76,119)
jnh302	100	900	63	**2**	(27,28) (98,208)
jnh310	100	900	184	**2**	(12,13) (90,188)
ezfact16_1	193	1113	203	**1**	(37,54)
ezfact16_2	193	1113	104	**1**	(32,41)
ezfact16_3	193	1113	207	**1**	(63,128)
3col40_5_3	80	346	4.64	**1**	(64,136)
fphp-012-010	120	1212	57	**1**	(120,670)

sistent cover enables us to get an lower-bound of the unsatisfied clauses in a max-SAT solution of an unsatisfiable instance. Indeed, the extracted MUSes do not share constraints and we know that all interpretations falsify at least one clause per MUS. Let us also note that inconsistent covers on `fpga` formulas suggest the presence of a form of symmetry in this type of instances.

8 Approximating the Set of All MUSes

Based on the `OMUS` procedure, we now address the problem of computing the set of MUSes of unsatisfiable instances, also called *clutter* by Bruni [6]. Since a MUS can be "broken" by removing one of its clauses, a naive approach consists in removing one clause of a MUS after this latter one has been discovered by the `OMUS` procedure, and then in iterating the process. Such an approach would deliver the right result when any pair of MUSes exhibits an empty intersection. However, MUSes can have non-empty intersections. Accordingly, when we remove a clause from a MUS, we actually break all MUSes containing it. To prevent this drawback from occurring as much as possible, we should prefer dropping clauses that belong to a minimal number of MUSes. Accordingly, we have investigated the following heuristic.

As max-SAT is intended to deliver a minimal number of unsatisfied clauses, the remaining unsatisfied clauses in a max-SAT solution must belong to intersections of MUSes as much as possible. Accordingly, for each clause, we record the minimum number of clauses that have been falsified at the same time during a failed LS. After a MUS is detected, the clause in the MUS with the highest score is removed from the

Algorithm 5: ASMUS algorithm

Input: an unsatisfiable SAT formula Σ
Output: an Approximation of the Set of MUSes of Σ

```
1  begin
2      S ⟵ ∅ ;
3      while Σ is unsatisfiable do
4          MUS ⟵ OMUS(Σ) ;
5          S ⟵ S ∪ {MUS} ;
6          c ⟵ a clause of MUS s.t. c has the highest score (in terms of
7               minimum of number of falsified clauses);
8          Σ ⟵ Σ \ {c} ;
9      return S ;
10 end
```

instance.

We call this approach ASMUS (Approximate Set of MUS) and describe it in Algorithm 5. Clearly, ASMUS is incomplete. However, it delivers very good results with respect to current existing approaches, as illustrated by our experimental investigations summarized in Table 4. For these experimentations, the time-out was set to 20000 s. AleatX_Y_Z instances are standard generated (unsatisfiable) random ones, with X variables and Y clauses. XAIMY_Z instances are the mere concatenations of X AIMα_β instances, where $\alpha = \frac{Y}{X}$ and $\beta = \frac{Z}{Y}$.

We have compared the ASMUS method with the complete algorithm proposed in [20] from an experimental point of view. Table 4 shows that both approaches appear to deliver the *exact* sets of MUSes on the simple "aim" benchmarks, using similar run-times. On more difficult instances like Aleat30_75_*, ASMUS almost extracts all MUSes and its computation time is in general better than the complete method one.

Moreover, Liffiton and Sakallah's algorithm can get into trouble for larger instances, since a formula can exhibit an exponential number of MUSes and since their approach aims at computing all MUSes individually, only after having computed all maximally satisfiable subformulas, which can be intractable. On the opposite, our approximation technique does not suffer from such a drawback and exhibits an anytime property since MUSes are directly computed one after the other. For example, let us consider the `Aleat20_70_2` random instance. It exhibits 70 clauses and these constraints form more than 114 000 MUSes. Due to the very small size of this formula, [20] has been able to compute all MUSes. For larger instances involving many MUSes, like `dp02u01` (213 atoms, 376 clauses), the set of MUSes could not be computed within 20000 seconds, while our approach extracted 14 MUSes in 26 seconds.

9 Conclusions

In this paper, we have introduced a novel local search counting heuristic that proves useful for extracting minimal explanations of unsatisfiability of SAT instances. It is

Table 4: Finding as many MUSes as possible.

Instance	#var	#cla	L.&S.		ASMUS	
			#MUS	Temps	#MUS	Temps
aim100-1_6-no-1	100	160	1	0.18	1	0.31
aim200-1_6-no-1	200	320	1	0.14	1	0.68
aim200-1_6-no-2	200	320	2	0.22	2	0.76
aim200-2_0-no-3	200	400	1	0.12	1	0.56
aim200-2_0-no-4	200	400	2	0.26	2	0.88
Aleat20_70_1	20	70	127510	6.9	6	4.9
Aleat20_70_2	20	70	114948	10.8	13	8.7
Aleat30_75_1	30	75	11	59.82	7	2.2
Aleat30_75_2	30	75	9	26.84	8	2.9
Aleat30_75_3	30	75	10	12.84	10	3.7
Aleat50_218_1000	50	218	Time out		67	173
Aleat50_218_100	50	218	Time out		39	126
2AIM100_160	100	160	2	0.21	2	0.69
2AIM400_640	400	640	2	14.9	2	3.1
3AIM150_240	150	240	3	73.84	3	1.46
4AIM200_320	200	320	Time out		4	2.82
dp02u01	213	376	Time out		14	26.12
Homer06	180	830	Time out		2	17.47

based on a concept of critical clause, which sheds light on several forms of falsified clauses during a local search run. Roughly, a critical clause is a falsified clause that can be satisfied thanks to a local search flip only when some other clauses become falsified at the same time. To some extent, a clause that is falsified but that is not critical is thus a clause that does not really participate in the infeasibility of the instance, since it can be satisfied without leading other clauses to become $false$ in their turn. The main properties of this criticial clause concept have been analyzed formally. We have shown that it plays a crucial role in a new heuristic-based approach to approximate or compute MUSes. Accordingly, our experimental results show the very good performance of this new approach that tracks MUSes according to the trace of a failed local search run for satisfiability checking. We have also introduced a strict inconsistent cover concept. This concept allows us to avoid computing all MUSes when the goal is to explain enough infeasibility causes that would allow the instance to regain satisfiability if they were all repaired. Accordingly, our heuristic-based approach has been extended to address the search for inconsistent covers. Once again, the approach proves efficient on many difficult benchmarks. In the future, we plan to explore how the critical clause concept could be refined in order to further improve the performance of our algorithms.

References

[1] J. Bailey and P. J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *International Symposium on Practical Aspects of Declarative Languages (PADL'05)*, pages 174–186, 2005.

[2] H.K. Büning. On subclasses of minimal unsatisfiable formulas. *Discrete Applied Mathematics*, 107(1–3):83–98, 2000.

[3] F. Boussemart, F. Hémery, C. Lecoutre, and L. Saïs. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence (ECAI'04)*, pages 146–150, 2004.

[4] L. Brisoux, E. Grégoire, and L. Saïs. Checking depth-limited consistency and inconsistency in knowledge-based systems. *International Journal of Intelligent Systems*, 16(3):333–360, 2001.

[5] R. Bruni. Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Applied Mathematics*, 130(2):85–100, 2003.

[6] R. Bruni. On exact selection of minimally unsatisfiable subformulae. *Annals of Mathematics and Artificial Intelligence*, 43(1):35–50, 2005.

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5(7):394–397, July 1962.

[8] 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.

[9] DIMACS. Benchmarks on SAT. ftp://dimacs.rutgers.edu/pub/challenge/-satisfiability/.

[10] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactual. *Artificial Intelligence*, 57:227–270, 1992.

[11] 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.

[12] E. Gregoire, B. Mazure, and C. Piette. Boosting a complete technique to find MSSes and MUSes thanks to a local search oracle. In *International Joint Conference on Artificial Intelligence (IJCAI'07)*, volume 2, pages 2300–2305, 2007.

[13] E. Grégoire and D. Ansart. Overcoming the Christmas tree syndrome. *International Journal on Artificial Intelligence Tools (IJAIT)*, 9(2):97–111, 2000.

[14] E. Grégoire, B. Mazure, and L. Saïs. Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently. In *International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'02)*, number 2443 in LNCS, pages 51–60. Springer, 2002.

[15] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[16] J. Huang. MUP: A minimal unsatisfiability prover. In *Asia and South Pacific Design Automation Conference (ASP-DAC'05)*, pages 432–437, 2005.

[17] U Junker. QuickXPlain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, pages 75–82, August 2001.

[18] H. Kautz, B. Selman, and D. McAllester. Walksat in the SAT 2004 competition. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[19] O. Kullmann, I. Lynce, and J. P. Marques Silva. Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 22–35, 2006.

[20] M.H. Liffiton and K.A. Sakallah. On finding all minimally unsatisfiable subformulas. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 173–186, 2005.

[21] I. Lynce and J. Marques-Silva. On computing minimum unsatisfiable cores. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[22] B. Mazure, L. Saïs, and E. Grégoire. A powerful heuristic to locate inconsistent kernels in knowledge-based systems. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'96)*, pages 1265–1269, 1996.

[23] B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, 22:319–322, 1998.

[24] M. N. Mneimneh, I. Lynce, Z. S. Andraus, J. P. Marques Silva, and K. A. Sakallah. A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 467–474, 2005.

[25] Y. Oh, M.N. Mneimneh, Z.S. Andraus, K.A. Sakallah, and I.L. Markov. AMUSE: a minimally-unsatisfiable subformula extractor. In *Design Automation Conference (DAC'04)*, pages 518–523, 2004.

[26] C.H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

[27] SATLIB. Benchmarks on SAT. http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html.

[28] L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, 2003.

Using local search to find MSSes and MUSes

Article publié dans la revue « *European Journal of Operational Research* » volume 199, n°3, pages 640-646, 2009.

Co-écrit avec Éric GRÉGOIRE et Cédric PIETTE.

Using Local Search to find MSSes and MUSes

Éric Grégoire, Bertrand Mazure and Cédric Piette

Université Lille-Nord de France, Artois, F-62307 Lens
CRIL, F-62307 Lens
CNRS UMR 8188, F-62307 Lens
rue Jean Souvraz SP18 F-62307 Lens France

Abstract

In this paper, a new complete technique to compute Maximal Satisfiable Subsets (MSSes) and Minimally Unsatisfiable Subformulas (MUSes) of sets of Boolean clauses is introduced. The approach improves the currently most efficient complete technique in several ways. It makes use of the powerful concept of critical clause and of a computationally inexpensive local search oracle to boost an exhaustive algorithm proposed by Liffiton and Sakallah. These features can allow exponential efficiency gains to be obtained. Accordingly, experimental studies show that this new approach outperforms the best current existing exhaustive ones.

Key words: SAT, MSSes, MUSes, satisfiability, hybrid algorithm

1 Introduction

This last decade, the SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, has received much attention from the AI research community. Indeed, SAT appears to be a cornerstone in many domains, like e.g. nonmonotonic reasoning, automated reasoning, model-based diagnosis, planning and knowledge bases verification and validation. However, only knowing that a SAT instance is unsatisfiable is often not satisfactory since we might prefer knowing what goes *wrong* with the instance when this latter one is expected to be satisfiable.

In this respect, the MUS (*Minimally Unsatisfiable Subformula*) concept can be crucial since a MUS can be seen as an irreducible cause for infeasibility. Indeed, a

Email addresses: gregoire@cril.fr (Éric Grégoire), mazure@cril.fr (Bertrand Mazure), piette@cril.fr (Cédric Piette).

MUS is an unsatisfiable set of clauses such that any of its subsets is satisfiable. It thus provides one explanation for unsatisfiability that cannot be made shorter in terms of the number of involved clauses. Restoring the satisfiability of an instance cannot be done without fixing all its MUSes.

Unfortunately, a same instance can exhibit several MUSes. Actually, the number of these MUSes can be exponential since a n-clauses SAT instance can exhibit $C_n^{n/2}$ MUSes in the worst case. Moreover, computing MUSes is intractable in the general case. Indeed, checking whether a set of clauses is a MUS or not is DP-complete [1] and checking whether a formula belongs to the set (clutter) of MUSes of an unsatisfiable SAT instance or not, is in Σ_2^p [2]. Fortunately, the number of MUSes remains often tractable in real-life applications. For example, in model-based diagnosis [3], it is often assumed that single faults occur most often, which can entail small numbers of MUSes.

A dual concept is the notion of *Maximal Satisfiable Subset* (MSS) of a SAT instance, and the complement of a MSS in a SAT instance is called a CoMSS. The complete set of MUSes or MSSes is an implicit encoding of the other [4]. Specifically, a CoMSS is a hitting set of the set of MUSes and represent minimal sets of clauses that should be dropped in order to restore consistency. In this paper, we are interested in exhaustive approaches to compute these three correlated concepts in the full Boolean clausal framework.

Recently, several approaches have been proposed to approximate or compute MUSes and MSSes, both in the Boolean framework and for other types of constraints. Some of them concern specific classes of clauses or remain tractable for small instances, only. Among them, let us mention the approach in [5], where it is shown how a MUS can be extracted in polynomial time through linear programming techniques for clauses exhibiting a so-called integral property. However, only restrictive classes of clauses obey such a property (mainly Horn, renamable Horn, extended Horn, balanced and matched ones). Let us also mention [6][7][8], which contain studies of the complexity and the algorithmic aspects of extracting MUSes for specific classes of clauses. In [9], an approach is proposed that approximates MUSes by means of an adaptative search guided by clauses hardness. In [10] a technique is described, that extracts MUSes by learning nogoods involved in the derivation of the empty clause by resolution. In [11], a complete and exhaustive technique to extract smallest MUSes is introduced. In [12], a DPLL-oriented approach has been presented that is based on a marked clauses concept to allow one to approximate MUSes. In [13], Grégoire, Mazure and Piette have proposed a heuristic-based incomplete approach to compute MUSes, which outperforms competing ones from a computational point of view.

Interestingly, in [14] the same authors have introduced a concept of inconsistent covers to circumvent the possible intractable number of MUSes, and presented a technique to compute them. Roughly, an inconsistent cover of an unsatisfiable

SAT instance represents a set of MUSes that covers enough independent causes of inconsistency that would allow the instance to regain consistency if they were repaired. Although an inconsistent cover does not provide us with the set of all MUSes that may be present in a formula, it does however provide us with a series of minimal explanations of inconsistency that are sufficient to explain and potentially "fix" enough causes of inconsistency in order for the whole instance to regain consistency.

These latter techniques are incomplete ones in the sense that they do not necessarily deliver all MUSes. However, in some application domains, it can be necessary to find the set of *all* MUSes, because diagnosing infeasibility is hard, if not impossible, without a complete view of its causes [4]. Obviously enough, such techniques can only remain tractable provided that the number of MUSes remains itself tractable. Likewise, the number of MSSes and CoMSSes can be exponential in the worst case. It should be noted that many domains in Artificial Intelligence like belief revision (see e.g. [15]) involve conceptual approaches to handle unsatisfiability that can require the complete sets of MUSes, MSSes, and CoMSSes to be computed in the worst case, even when additional epistemological ingredients like e.g. stratification are introduced in the logical framework.

In this paper, the focus is on complete techniques. We introduce a new complete technique to compute all MUSes, MSSes and CoMSSes of a SAT instance, provided obvious tractability limitations. It improves the currently most efficient complete technique, namely Liffiton and Sakallah'one [4] (in short L&S), which in turn was shown more competitive than previous approaches by Bailey and Stuckey [16], and by de la Banda, Stuckey and Wazny [17], which were introduced in somewhat different contexts.

Our approach exhibits two main features. First, it is a hybridization of the L&S complete approach with a local search pretreatment. A local search technique is indeed used as an oracle to find potential CoMSSes of the SAT instance, which are themselves hitting sets of MUSes. We show that such a hybridization can yield exponential efficiency gains. Second, the efficiency of the approach relies on the crucial concept of critical clause, which appears to be a powerful ingredient of our technique to locate MUSes.

The rest of the paper is organized as follows. First, the reader is provided with the necessary background about SAT, MUSes and the dual concepts of MSSes and CoMSSes. Then, Liffiton and Sakallah's exhaustive approach is briefly presented. In Section 4, we show how this technique can be valuably hybridized with a local search pretreatment, making use of the critical clause concept. It is shown how this pretreatment can be theoretically valuable from a computational point of view. In Section 5, we compare this new approach with Liffiton and Sakallah's one on various benchmarks.

2 Background

In this section, the reader is provided with basic notions about SAT, MUSes, MSSes and CoMSSes.

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables, noted a, b, c, etc. The $\wedge$, $\vee$, $\neg$ and $\Rightarrow$ symbols represent the standard conjunctive, disjunctive, negation and material implication connectives, respectively. Formulas and clauses will be noted using upper-case letters such as C. Sets of formulas will be represented using Greek letters like Γ or Σ. An interpretation is a truth assignment function that assigns values from $\{$*true*, *false*$\}$ to every Boolean variable. A formula is satisfiable when there is at least one interpretation (called model) that satisfies it, i.e. that makes it become *true*. An interpretation will be noted by upper-case letters like I and will be represented by the set of literals that it satisfies. Actually, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) using a set (interpreted as a conjunction) of clauses, where a clause is a finite disjunction of literals, where a literal is a Boolean variable that is possibly negated. SAT is the NP-complete problem that consists in checking whether a set of Boolean clauses is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not.

When a SAT instance is unsatisfiable, it exhibits at least one *Minimally Unsatisfiable Subformula*, in short one *MUS*.

Definition 1 *A MUS Γ of a SAT instance Σ is a set of clauses s.t.*

(1) $\Gamma \subseteq \Sigma$
(2) Γ *is unsatisfiable*
(3) $\forall \Delta \subset \Gamma$, Δ *is satisfiable*

Example 2 *Let* $\Sigma = \{a, \neg c, \neg b \vee \neg a, b, \neg b \vee c\}$. Σ *exhibits two MUSes, namely* $\{a, b, \neg b \vee \neg a\}$ *and* $\{\neg c, b, \neg b \vee c\}$.

A dual concept is the notion of *Maximal Satisfiable Subset* (MSS) of a SAT instance.

Definition 3 *A MSS Γ of a SAT instance Σ is a set of clauses s.t.*

(1) $\Gamma \subseteq \Sigma$
(2) Γ *is satisfiable*
(3) $\forall \Delta \subseteq (\Sigma \setminus \Gamma)$ *s.t.* $\Delta \neq \emptyset$, $\Gamma \cup \Delta$ *is unsatisfiable.*

The set-theoretical complement of a MSS w.r.t. a SAT instance is called a *CoMSS*.

Definition 4 *The CoMSS of a MSS Γ of a SAT instance Σ is given by* $\Sigma \setminus \Gamma$

Example 5 *Let us consider the formula Σ from the previous example. Σ exhibits five MSSes:* $\{b\}$, $\{\neg c, a\}$, $\{\neg c, \neg b \vee c\}$, $\{\neg b \vee c, \neg b \vee \neg a\}$, $\{\neg c, \neg b \vee \neg a\}$

As shown by several authors [4], these concepts are correlated. Mainly, a CoMSS contains at least one clause from each MUS. Actually, a CoMSS is an irreducible hitting set of the set of MUSes. In a dual way, every MUS of a SAT instance is an irreducible hitting set of the CoMSSes. Accordingly, as emphasized by [4], although `MINIMAL-HITTING-SET` is a NP-hard problem, irreducibility is a less strict requirement than minimal cardinality. Actually, a MUS can be generated in polynomial time from the set of CoMSSes.

3 Liffiton and Sakallah's Exhaustive Approach

Liffiton and Sakallah's approach [4] to compute all MUSes (in short L&S) is based on the strong duality between MUSes and MSSes. To the best of our knowledge, it is currently the most efficient one. First it computes all MSSes before it extracts the corresponding set of MUSes. Here, the focus is on L&S first step since we shall improve it and adopt the second step as such.

L&S is integrated with a modern SAT solver and takes advantage of it. Roughly, every ith clause $C_i = x_1 \vee ... \vee x_m$ of the SAT instance is augmented with a negated clause selector variable y_i to yield $C'_i = x_1 \vee ... \vee x_m \vee \neg y_i$. While solving these new clauses, assigning y_i to *false* has the effect of disabling or removing C_i from the instance. Accordingly, a MSS can be obtained by finding a satisfying assignment with a minimal number of y_i variables assigned *false*. The algorithm makes use of a sliding objective approach allowing for an incremental search. A bound on the number of y_i that may be assigned to *false* is set. For each value of the bound, starting at 0 and incrementing by 1, an exhaustive search is performed for all satisfiable assignments to the augmented formula C'_i, which will find all CoMSSes having their size equal to the bound. Whenever one solution is found, it is recorded, and a corresponding clause forcing out that solution (and any supersets of it) is inserted. This blocking clause is a disjunction of the y_i variables for the clauses in that CoMSS.

Before beginning the search with the next bound, the algorithm checks that the new instance augmented with all the blocking clauses is still satisfiable without any bound on the y_i variables. If there is no such satisfying assignment, the entire set of CoMSSes has been found and the algorithm terminates.

The second part of the algorithm computes the complete set of MUSes from the set of CoMSSes in a direct way. The approach that we shall introduce will include this second step as such.

4 Local Search and Critical Clauses

In this section, it is shown how the aforementioned exhaustive search algorithm can be improved in a dramatic manner by hybridizing it with an initial local search step, which provides valuable oracles for the subsequent exhaustive search process. We shall call the new approach HYCAM (HYbridization for Computing All Muses).

First, let us motivate our approach in an intuitive manner. Clearly, a (fast) initial local search run for satisfiability on the initial instance might encounter some actual MSSes. Whenever this phenomenon happens, it can prove valuable to record the corresponding CoMSSes in order to avoid computing them during the subsequent exhaustive search. Moreover, rather than checking whether we are faced with an actual MSS or not, it can prove useful to record the corresponding candidate CoMSS that will be checked later during the exhaustive search. Obviously enough, we must study which interpretations encountered during the local search process yield candidate MSSes and criteria must be defined in order to record a limited number of potentially candidate CoMSSes only. In this respect, a concept of critical clause will prove extremely valuable in the sense that it allows us to state necessary conditions for being a CoMSS that can be checked quickly. When all the remaining candidate CoMSSes are recorded, the incremental approach by Liffiton and Sakallah allows us to exploit this information in a very valuable and efficient way. Let us describe this in a more detailed manner.

A local search algorithm is thus run on the initial SAT instance. The goal is to record as many candidate CoMSSes as possible, based on the intuitive heuristics that local search often converges towards local minima, which could translate possibly good approximations of MSSes. A straightforward approach would consist in recording for each visited interpretation the set of unsatisfied clauses. Obviously enough, we do not need to record supersets of already recorded candidate CoMSSes since they cannot be actual CoMSSes as they are not minimal with respect to set-theoretic inclusion. More generally, we have adapted the technique proposed by Zhang in [18] to sets of clauses in order to record the currently smaller candidates CoMSSes among the already encountered series of sets of unsatisfied clauses. Now, crucial ingredients in our approach are the concepts of once-satisfied and critical clauses. Moreover, we have also exploited the following concept of critical clause, which has already proved valuable for locating MUSes and inconsistent covers using an incomplete technique based on local search [13][14].

Definition 6 *A clause C is once-satisfied by an interpretation I iff exactly one literal of C is satisfied by I. A clause C that is falsified by the interpretation I is critical w.r.t. I iff the opposite of each literal of C belongs to at least one once-satisfied clause by I.*

Intuitively, a critical clause is thus a falsified clause that requires at least another

Algorithm 1: Local Search approximation

Input: a CNF formula Σ
Output: Set of candidate CoMSS
begin
  $candidates \longleftarrow \emptyset$;
  $\#fail \longleftarrow 0$;
  $I \longleftarrow generate\_random\_interpretation()$;
  while ($\#fail <$ PRESET_#FAILURES_AUTHORIZED) **do**
    $newcandidates \longleftarrow$ FALSE ;
    for $j = 1$ **to** #FLIPS **do**
      Let Δ be the set of falsified clauses by I ;
      if $\forall C \in \Delta$, C *is critical* **and** Δ *is not implied in* $candidates$ **then**
        removeAllSetImplied(Δ,$candidates$) ;
        $candidates \longleftarrow \Delta \cup candidates$;
        $newcandidates \longleftarrow$ TRUE ;
      flip(I) ;
    if $not(newcandidates)$ **then** $\#fail \longleftarrow \#fail + 1$;
  return $candidates$;
end

clause to be falsified in order to become satisfied, performing a flip. The following proposition shows how this concept allows us to eliminate wrong candidates CoMSSes.

Proposition 7 *Let Σ be a SAT instance and let I be an interpretation. Let Γ be a non-empty subset of Σ s.t. all clauses of Γ are all falsified by I. When at least one clause of Γ is not critical w.r.t. I, then Γ is not a CoMSS of Σ.*

PROOF. By definition, when a clause C_f of Γ is not critical w.r.t. I, there exists at least one literal $c \in C_f$ whose truth-value can be inversed (i.e. flipped) without falsifying any other clause of Σ. Accordingly, Γ is not minimal and cannot be a CoMSS of Σ. □

In practice, testing whether all falsified clauses are critical or not can be performed quickly and prevents many sets of clauses to be recorded as candidate CoMSSes.

Using these features, the local search run on the initial SAT instance, as described in Algorithm 1, yields a series of candidate CoMSSes. This information proves valuable and allows us to boost L&S complete search.

L&S is incremental in the sense that it computes CoMSSes of increasing sizes, progressively. After n iterations have been performed, all CoMSSes of cardinality lower or equal than n have been obtained. Accordingly, if we have recorded candidate CoMSSes containing $n + 1$ clauses, and if they are not supersets of already

obtained CoMSSes, we are sure that they are actual CoMSSes. In this respect, we do not need to search them, and their corresponding blocking clauses can be inserted directly. Moreover, we do not need to perform the SAT test at the end of the n-th iteration, since we are then aware of the existence of larger CoMSSes.

It is also easy to show that the insertion of these blocking clauses can allow both NP-complete and CoNP-complete tests to be avoided. Let us illustrate this on an example.

Example 8 *Let Σ be the following SAT instance.*

$$\Sigma = \begin{cases} c_0 : (d) \\ c_1 : (b \vee c) \\ c_2 : (a \vee b) \\ c_3 : (a \vee \neg c) \\ c_4 : (\neg b \vee \neg e) \\ c_5 : (\neg a \vee \neg b) \\ c_6 : (a \vee e) \\ c_7 : (\neg a \vee \neg e) \\ c_8 : (b \vee e) \\ c_9 : (\neg a \vee b \vee \neg c) \\ c_{10} : (\neg a \vee b \vee \neg d) \\ c_{11} : (a \vee \neg b \vee c) \\ c_{12} : (a \vee \neg b \vee \neg d) \end{cases}$$

Σ is an unsatisfiable SAT instance made of 13 clauses and making use of 5 variables. It exhibits 3 MUSes, which are illustrated in Figure 1, and admits 19 MSSes. Assume that both L&S and HYCAM are run on this instance. Its clauses are augmented by the $\neg y_i$ negated clause selector variables. Assume also that the local search performed by HYCAM provides 4 candidate CoMSSes: $\{c_5\}$, $\{c_3, c_2\}$, $\{c_0, c_1, c_2\}$ and $\{c_3, c_8, c_{10}\}$.

If the branching variables are chosen based on the lexical order, then a and b are assigned to $true$ and c_5 is falsified. Thus, L&S tries to prove that this clause forms a CoMSS, which requires a NP-complete test (because it has to find a model of $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b\}$). On the contrary, when HYCAM is run, the blocking clause y_5 is added before the first iteration of the complete algorithm is performed, since y_5 is the clause selector variable of c_5 and since the local search has already delivered

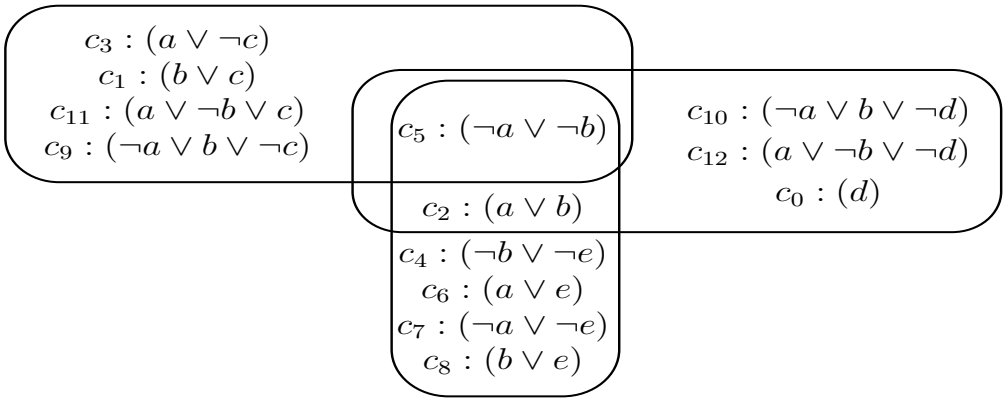


Fig. 1. MUSes of Example 8

this CoMSS. In consequence, when a and b are assigned $true$, the DPLL-algorithm backtracks immediately as the $\{y_5, \neg y_5\}$ unsatisfiable set has been obtained, without requiring any further NP-complete test.

Similarly, the introduction of additional clause selector variables by HYCAM can reduce the number of CoNP-complete tests. For example, let us assume that e is the first branching variable, that e is assigned $false$ and that the next variables are selected according to the lexical order. When a and b are assigned $true$, L&S tries to prove that $\{c_5\}$ is a CoMSS. Since $\neg e$ is tautological consequence of $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b\}$, no model exists for $\Sigma \backslash \{\neg a \vee \neg b\} \cup \{a, b, \neg e\}$. Clearly, such a test is in CoNP. Thanks to the previously delivered candidate CoMSSes, HYCAM avoids this part of the search space to be explored. Indeed, since we know that $\{c_5\}$ is a CoMSS, when a and b are assigned $true$, no further CoNP tests are performed with respect to this partial assignment.

In fact, from a computational point of vue, the preliminary non-expensive local search eliminates nodes in the search tree, avoiding both NP and CoNP tests.

The HYCAM algorithm is depicted in Algorithm 2.

5 Experimental Evaluation

HYCAM has been implemented and compared to L&S from a practical point of view. For both algorithms, the complete search step is based on the use of MiniSat [19], which is currently one of the best modern SAT solvers. As a case study, we used Walksat [20] for the local search pretreatment. The number of flips and tries of Walksat is related to the number of candidate CoMSS already found. For each try, a small number of flips is performed. If no new candidate is found during a try then a counter is incremented. When this counter exceeds a threshold (experimentally

Algorithm 2: The HYCAM algorithm

Input: a CNF formula Σ
Output: All MSSes of Σ
begin
  $cand \longleftarrow$ LS_approximation(Σ) ; /* algorithm 1 */
  $\Sigma_y \longleftarrow$ addSelectorClauses(Σ) ;
  $k \longleftarrow 0$;
  $MSS \longleftarrow \emptyset$;
  while *SAT(Σ_y)* **do**
    removeAllSetImplied($\{\Sigma \backslash C | C \in MSS\}$,$cand$) ;
    $\Sigma_y \longleftarrow$ addBlockingClausesOfSize(k,$cand$) ;
    $MSS \longleftarrow MSS \cup \{\Sigma \backslash C | C \in cand$ and $|C| = k\}$;
    $MSS \longleftarrow MSS \cup$ SAT_with_bound(k,Σ_y) ;
    $k \longleftarrow k + 1$;
  return MSS ;
end

set to 30), we consider that no new candidate could be found by the local search. This way to end the local search pretreatment offers a good trade-off between the number of candidates found and the time spent. Besides, for all experiments, the time consumed by the local search step was less than 5% of the global time. All our experimental studies have been conducted on Intel Xeon 3GHz under Linux CentOS 4.1. (kernel 2.6.9) with a RAM memory size of 2Go. In the following, a time-out limit has been set to 3 CPU hours.

First, in Table 1, we report experimental results about the computation of MSSes on pigeon-hole and xor-chains benchmarks [21], which are globally unsatisfiable in the sense that removing any one of their clauses makes the instance become satisfiable. Obviously enough, such instances exhibit a number of CoMSSes equals to their number of clauses, and the size of any of their CoMSS is one. A significant time gap can be observed in favor of HYCAM. The efficiency gain ratio is even more significant when the size of the instance increased. For these instances, the local search run often succeeds in finding all CoMSSes, and the complete step often reduces to an unsatisfiability test. On the contrary, L&S explores many more nodes in the search space to deliver the CoMSSes.

In Table 2, experimental results on more difficult benchmarks from the annual SAT competition [21] are described. Their number of MSSes is often exponential, and computing them often remains intractable. Accordingly, we have limited the search to CoMSSes of restricted sizes, namely we have set a size limit to 5 clauses. As our experimental results illustrate, HYCAM outperforms L&S. For example, let us consider `rand_net40-30-10`. This instance contains 5831 MSSes (with the size of their corresponding CoMSSes less than 5). L&S and HYCAM deliver this result in 1748 and 386 seconds, respectively. For the `ca256` instance, HYCAM has extracted 9882 MSSes in less than 5 minutes whereas L&S did not manage to

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

Table 1
L&S vs. HYCAM on globally unsatisfiable instances

produce this result within 3 hours. Let us note that HYCAM also delivers CoMSSes made of 5 clauses after its computation is ended since we know that all sets of 5 falsified clauses recorded by the local search run and that are not supersets of the obtained smaller CoMSSs are actually also CoMSSes.

In Table 3, experimental results on hard instances to compute the complete set of MSSes and MUSes are reported. As explained above, both L&S and HYCAM approaches require all MSSes to be obtained before MUSes are computed. By allowing complete sets of MSSes to be delivered in a shorter time, HYCAM allows the complete set of MUSes to be computed for more instances and in a faster manner than L&S does. Obviously enough, when the number of MSSes or MUSes are exponential, computing and enumerating all of them remain intractable.

For instance, L&S was unable to compute all MSSes of the `php-012-011` instance within 3 hours CPU time, and could thus not discover its single MUS. HYCAM extracted it in 2597 seconds. On all instances exhibiting unique or a non-exponential number of MUSes, HYCAM was clearly more efficient than L&S. For example, on the `dlx2_aa` instance, L&S and HYCAM discovered the 32 MUSes within 3.12 and 0.94 seconds, respectively. Let us note that the additional time spent to compute all MUSes from the set of MSSes is often very small unless of course the number of MUSes is exponential.

Instance	**(#var,#cla)**	**#MSSes**	**L&S** (sec.)	**HYCAM** (sec.)
rand_net40-25-10	(2000,5921)	5123	893	197
rand_net40-25-5	(2000,5921)	4841	650	174
rand_net40-30-10	(2400,7121)	5831	1748	386
rand_net40-30-1	(2400,7121)	7291	1590	1325
rand_net40-30-5	(2400,7121)	5673	2145	402
ca032	(558,1606)	1173	4	1
ca064	(1132,3264)	2412	59	3
ca128	(2282,6586)	4899	691	18
ca256	(4584,13236)	9882	*time out*	277
2pipe	(892,6695)	3571	130	36
2pipe_1_ooo	(834,7026)	3679	52	30
2pipe_2_ooo	(925,8213)	5073	148	61
3pipe_1_ooo	(2223,26561)	17359	5153	2487
am_5_5	(1076,3677)	1959	68	57
c432	(389,1115)	1023	4	1
c880	(957,2590)	2408	28	3
bf0432-007	(1040,3668)	10958	233	98
velev-sss-1.0-cl	(1453,12531)	4398	1205	513

Table 2
L&S vs. HYCAM on various difficult SAT instances

6 Conclusions and Future Research

Computing all MSSes, CoMSSes and MUSes are highly intractable issues in the worst case. However, it can make sense to attempt to compute them for some real-life applications. In this paper, we have improved the currently most efficient exhaustive technique, namely Liffiton and Sakallah's method, in several ways. Our experimental results show dramatic efficiency gains for MSSes, CoMSSes and MUSes extracting. One interesting feature of the approach lies in its anytime character for computing MSSes. MSSes of increasing sizes are computed gradually. Accordingly, we can put a bound on the maximum size of the CoMSSes to be extracted, limiting the computing resources needed to extract them. To some extent, both L&S and HYCAM prove more adapted to extract complete sets of MSSes and CoMSSes than complete sets of MUSes. Indeed, the procedure involves computing MSSes (and thus CoMSSes) first. In this respect, we agree with Liffiton

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

Table 3
L&S vs. HYCAM on computing all MUSes

and Sakallah that an interesting path for future research concerns the study of how MUSes could be computed progressively from the growing set of extracted MSSes.

Many artificial intelligence research areas have studied various problems involving the manipulation of MUSes, MSSes and CoMSSes, like model-based diagnosis, belief revision, inconsistency handling in knowledge and belief bases, etc. These studies are often conducted from a conceptual point of view, or from a worst-case complexity point of view, only. We believe that the practical computational progresses as such as the ones obtained in this paper can prove valuable in handling these problems practically. In this respect, future research could concentrate on deriving specific algorithms for these AI issues, exploiting results like the ones described in this paper.

Acknowledgments

We thank Mark Liffiton for making his system available to us. This work has been supported in part by the *Région Nord/Pas-de-Calais*. A preliminary version of this paper was published in the proceedings of IJCAI'07.

References

[1] C. H. Papadimitriou, D. Wolfe, The complexity of facets resolved, Journal of Computer and System Sciences 37 (1) (1988) 2–13.

[2] T. Eiter, G. Gottlob, On the complexity of propositional knowledge base revision, updates and counterfactual, Artificial Intelligence 57 (1992) 227–270.

[3] W. Hamscher, L. Console, J. de Kleer (Eds.), Readings in Model-Based Diagnosis, Morgan Kaufmann, 1992.

[4] M. Liffiton, K. Sakallah, On finding all minimally unsatisfiable subformulas, in: Proceedings of SAT'05, 2005, pp. 173–186.

[5] R. Bruni, On exact selection of minimally unsatisfiable subformulae., Annals of Mathematics and Artificial Intelligence 43 (1) (2005) 35–50.

[6] H. Büning, On subclasses of minimal unsatisfiable formulas, Discrete Applied Mathematics 107 (1–3) (2000) 83–98.

[7] G. Davydov, I. Davydova, H. Büning, An efficient algorithm for the minimal unsatisfiability problem for a subclass of cnf, Annals of Mathematics and Artificial Intelligence 23 (3–4) (1998) 229–245.

[8] H. Fleischner, O. Kullman, S. Szeider, Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference, Theoretical Computer Science 289 (1) (2002) 503–516.

[9] R. Bruni, Approximating minimal unsatisfiable subformulae by means of adaptive core search, Discrete Applied Mathematics 130 (2) (2003) 85–100.

[10] L. Zhang, S. Malik, Extracting small unsatisfiable cores from unsatisfiable boolean formula, in: Sixth international conference on theory and applications of satisfiability testing (SAT'03), Portofino (Italy), 2003, pp. 239–249.

[11] I. Lynce, J. Marques-Silva, On computing minimum unsatisfiable cores, in: Seventh international conference on theory and applications of satisfiability testing (SAT'04), Vancouver, 2004, pp. 305–310.

[12] Y. Oh, M. Mneimneh, Z. Andraus, K. Sakallah, I. Markov, AMUSE: a minimally-unsatisfiable subformula extractor, in: Proceedings of the 41th Design Automation Confrence (DAC 2004), 2004, pp. 518–523.

[13] É. Grégoire, B. Mazure, C. Piette, Extracting MUS, in: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06), Riva del Garda, Italy, 2006, pp. 387–391.

[14] É. Grégoire, B. Mazure, C. Piette, Tracking mus and strict inconsistent cover, in: Proceedings of International Conference on Formal Methods in Computer-Aided Design (FMCAD'06), San Jose, CA, USA, 2006, pp. 39–46.

[15] B. Bessant, É. Grégoire, P. Marquis, L. Saïs, Iterated Syntax-Based Revision in a Nonmonotonic Setting, Vol. 22 of Applied Logic, Kluwer Academic Publishers, Applied Logic Series, 2001, Ch. of Frontiers in Belief Revision, pp. 369–391.

[16] J. Bailey, P. Stuckey, Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization, in: PADL, 2005, pp. 174–186.

[17] M. de la Banda, P. Stuckey, J. Wazny, Finding all minimal unsatisfiable subsets, in: Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDL 2003), 2003, pp. 32–43.

[18] 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), 2005, pp. 482–489.

[19] N. Eén, N. Sörensson, Minisat home page http://www.cs.chalmers.se/cs/research/formalmethods/minisat (2004).

[20] H. Kautz, B. Selman, D. McAllester, Walksat in the SAT'2004 competition, SAT Competition 2004 - solver description.

[21] SATLIB, Benchmarks on SAT http://www.intellektik.informatik.tu-darmstadt.de/satlib/benchm.html (2006).

MUST : Provide a Finer-Grained Explanation of Unsatisfiability

Article publié dans les actes de « *the Thirteenth International Conference on Principles and Practice of Constraint Programming* », lncs 4741, pages 317-331, 2007.

Co-écrit avec Éric GRÉGOIRE et Cédric PIETTE.

MUST: Provide a Finer-Grained Explanation of Unsatisfiability

Éric Grégoire, Bertrand Mazure, and Cédric Piette

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

Abstract. In this paper, a new form of explanation and recovery technique for the unsatisfiability of discrete CSPs is introduced. Whereas most approaches amount to providing users with a minimal number of constraints that should be dropped in order to recover satisfiability, a finer-grained alternative technique is introduced. It allows the user to reason both at the constraints and tuples levels by exhibiting both problematic constraints and tuples of values that would allow satisfiability to be recovered if they were not forbidden. To this end, the Minimal Set of Unsatisfiable Tuples (MUST) concept is introduced. Its formal relationships with Minimal Unsatisfiable Cores (MUCs) are investigated. Interestingly, a concept of shared forbidden tuples is derived. Allowing any such tuple makes the corresponding MUC become satisfiable. From a practical point of view, a two-step approach to the explanation and recovery of unsatisfiable CSPs is proposed. First, a recent approach proposed by Hemery *et al.*'s is used to locate a MUC. Second, a specific SAT encoding of a MUC allows MUSTs to be computed by taking advantage of the best current technique to locate Minimally Unsatisfiable Sub-formulas (MUSes) of Boolean formulas. Interestingly enough, shared tuples coincide with protected clauses, which are one of the keys to the efficiency of this SAT-related technique. Finally, the feasibility of the approach is illustrated through extensive experimental results.

Keywords: CSP, constraint networks, explanation, unsatisfiability, MUC, MUS, MUST.

1 Introduction

In this paper, we are concerned with unsatisfiable finite CSPs, namely finite Constraint Satisfaction Problems for which no solution exists. Recent approaches to explain such a form of unsatisfiability have been defined at the constraints level. For example, Hemery *et al.* [1] have proposed an approach called `DC(wcore)` to detect Minimally Unsatisfiable Cores (MUCs) of CSPs, i.e. unsatisfiable subsets of constraints of the initial CSP that are such that dropping any one the constraints allows the resulting subset to become satisfiable. In this paper, a finer-grained alternative technique is introduced. Indeed, dropping constraints can be too much destructive. Finer-grained information could be provided to the user, allowing him (her) not only to pinpoint the causes of

C. Bessiere (Ed.): CP 2007, LNCS 4741, pp. 317–331, 2007.

unsatisfiability at the constraints level, but also detect the tuples of values that are forbidden by the constraints and that would lead to satisfiability if they were allowed by those constraints.

In this respect, the contribution of this paper is twofold. On the one hand, a *Minimally Unsatisfiable Set of Tuples* (MUST) concept is introduced: it is aimed at encompassing the aforementioned notion of tuples that can allow satisfiability to be regained. The formal relationships between MUSTs and MUCs are investigated. Interestingly, a concept of shared forbidden tuples is derived. Allowing any shared tuple makes the corresponding MUC become satisfiable. On the other hand, a two-step approach to the explanation and recovery of unsatisfiable CSPs is proposed. A specific SAT encoding of a MUC allows MUSTs to be computed by taking advantage of the best current technique to locate Minimally Unsatisfiable Sub-formulas (MUSes) of Boolean formulas. Interestingly enough, shared tuples coincide with protected clauses [2], which are one of the keys to the efficiency of this SAT-related technique.

Accordingly, the paper is organized as follows. First, some basic definitions about CSPs and MUCs are provided. In section 3, MUSTs are introduced and linked to MUCs in section 4. Next, an original two-step approach to explain and recover from unsatisfiable CSPs is described. In section 5, a SAT-related approach to compute MUSTs and shared forbidden tuples of a MUC is introduced. The feasibility of the approach is illustrated through extensive experimental results in section 6. Section 7 compares the contribution presented in this paper with the current existing works. In the conclusion, some interesting paths for future research are described.

2 Background: CSPs and MUCs

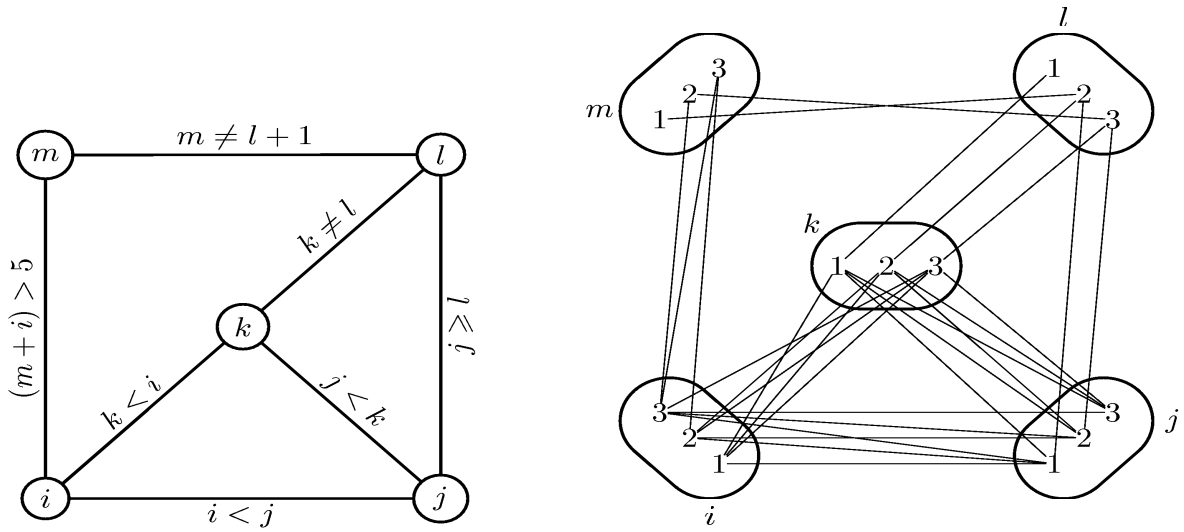
In this section, the reader is provided with basic concepts about CSPs and MUCs.

Definition 1. *A finite Constraint Satisfaction Problem (in short, CSP) is a pair* $P = \langle V,C \rangle$ *where*

1. V *is a finite set of* n *variables* $\{v_1, \ldots, v_n\}$ *s.t. each variable* $v_i \in V$ *has an associated finite instantiation domain, denoted* $dom(v_i)$*, which contains the set of possible values for* v_i*,*
2. C *is a finite set of* m *constraints* $\{c_1, \ldots, c_m\}$ *s.t. each constraint* $c_j \in C$ *involves a subset of variables of* V*, called scope and denoted* $Var(c_j)$*, and has an associated relation* $R(c_j)$*, which contains the set of tuples allowed for the variables of its scope.*

Definition 2. *Solving a CSP* $P = \langle V,C \rangle$ *consists in checking whether* P *admits at least one solution, i.e. an assignment of values for all variables of* V *s.t. all constraints of* C *are satisfied. If* P *admits at least one solution then* P *is called satisfiable else* P *is called unsatisfiable.*

In this paper, it will prove useful to adopt an alternative but equivalent definition for CSPs, expressed in terms of *forbidden* tuples of values.



(a) Constraint Network (b) Micro-structure

Fig. 1. Graph-representations of Example 1

Definition 3. *Let $\langle V, C\rangle$ be a CSP and let $c \in C$ s.t. $Var(c) = \{v_c^1, \ldots, v_c^l\}$. A forbidden tuple of values is a member of $dom(v_c^1) \times \cdots \times dom(v_c^l)$ s.t. $R(c)$ is not satisfied. The set of forbidden tuples of values for a constraint c is denoted $T(c)$.*

Accordingly, CSPs can be redefined as follows.

Definition 4. *A finite CSP is a pair $P = \langle V, C\rangle$ where*

1. *V is a finite set of n variables $\{v_1, \ldots, v_n\}$ s.t. each variable $v_i \in V$ has an associated finite instantiation domain, denoted $dom(v_i)$, which contains the set of possible values for v_i,*
2. *C is a finite set of m constraints $\{c_1, \ldots, c_m\}$ s.t. each constraint $c_j \in C$ involves a subset of variables of V, called scope and denoted $Var(c_j)$, and is given a relation $T(c_j)$, which contains the set of tuples forbidden for the variables of its scope.*

Accordingly, P will also be denoted $\langle V, \{(Var(c_1), T(c_1)),\ (Var(c_2), T(c_2)), \ldots, (Var(c_n), T(c_n))\}\rangle$.

Definition 5. *A constraint $c \in C$ of the CSP $P = \langle \{v_1, \ldots, v_n\}, C\rangle$ is falsified by an assignment $A \in dom(v_1) \times \cdots \times dom(v_n)$ iff the projection of A on $Var(c)$ is included in $T(c)$.*

In the following, *(forbidden) tuple* will be a shorthand for *forbidden tuple of values*, and binary CSPs will be considered only, namely CSPs where constraints involve two variables. Using binary CSPs do not restrict the impact of our works, since it is well-known that every discrete CSP can be reduced into a binary one, in polynomial time.

Example 1. Let V be {i,j,k,l,m} where each variable has the same domain {1,2,3}. Let C be a set of 7 constraints. In Figure 1(a), the CSP $P = \langle V, C\rangle$ is represented by

a so-called *constraint network*, namely a non-oriented graph, where each variable is a node and each constraint is an edge, labelled with its corresponding relation. It is also useful to represent a CSP by its *micro-structure*, which is a graph where the values of each variable are listed, and edges between values represent forbidden tuples. The micro-structure of this example is depicted in Figure 1(b).

When a CSP is infeasible, it exhibits at least one *Minimally Unsatisfiable Core*, or MUC. A MUC is a subpart of a CSP that is unsatisfiable and that does not contain any proper subpart that is also unsatisfiable.

Definition 6. *Let $P = \langle V, C\rangle$ and $P' = \langle V', C'\rangle$ be two CSPs. P' is an unsatisfiable core, in short a core, of P iff*

1. *P' is unsatisfiable*
2. *$V' \subseteq V$ and $C' \subseteq C$*

P' is a Minimal Unsatisfiable Core (MUC) of P iff

1. *P' is a core of P*
2. *there does not exist any proper core of P'*

Example 2. In the previous example, P is unsatisfiable. Indeed, P contains the MUC $P' = \langle V, \{i < j, j < k, k < i\}\rangle$: no assignment of values for i, j and k can be found such that these three constraints are satisfied, and dropping any constraint leads to satisfiability.

Computing one MUC for an unsatisfiable CSP is an NP-hard problem. More precisely, checking whether a constraint belongs to a MUC or not is in Σ_2^p [3]. Moreover, the number of MUCs inside a CSP can be exponential in the worst-case; it is in $\mathcal{O}(C_m^{m/2})$, where m is the number of constraints in the CSP. It should be noted that MUCs can share non-empty intersections. Several techniques have been proposed in the literature to compute MUCs, the `DC(wcore)` approach introduced recently by Hemery *et al.* [1] is claimed by its authors to be the most efficient one, most often.

3 MUSTs

Restoring the satisfiability of a CSP can be achieved through restoring the satisfiability of each of its MUCs. A natural way to break the unsatisfiability of a MUC is to drop any of its constraints. However, dropping some constraints as a whole can appear too much destructive. On the contrary, we might prefer to *weaken* one or several constraints, instead of removing them. One way to do that is to provide the user with some forbidden tuples that should be allowed in order to recover satisfiability.

Let us introduce the following MUST concept and study to which extent it can be a first good step in that direction. A MUST (*Minimally Unsatisfiable Set of Tuples*) of an unsatisfiable CSP P is an unsatisfiable CSP P' that is such that the sets of forbidden tuples of its constraints are subsets of the corresponding sets w.r.t. P and such that allowing any of the tuples of P' will render P' satisfiable.

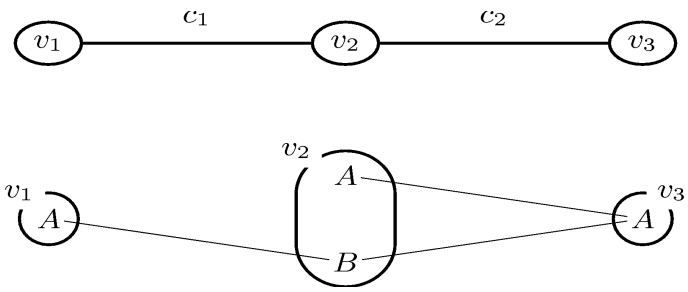


Fig. 2. Graphical representations of Example 3

Definition 7. *Let* $P = \langle V, \{(Var(c_1), T(c_1)), \ldots, (Var(c_m), T(c_m))\}\rangle$ *be an unsatisfiable CSP. The CSP* $P' = \langle V, \{(Var(c_1), T'(c_1)), \ldots, (Var(c_m), T'(c_m))\}\}\rangle$ *is a* MUST (Minimally Unsatisfiable Set of Tuples) *of* P *if and only if :*

1. P' *is unsatisfiable*
2. $\forall i$ *s.t.* $1 \leq i \leq m$, $T'(c_i) \subseteq T(c_i)$
3. $\forall i$ *s.t.* $1 \leq i \leq m$, $\forall\, T''(c_i) \subset T'(c_i)$, $\langle V, \{(Var(c_1), T'(c_1)), ..., (Var(c_i), T''(c_i)), ..., (Var(c_m), T'(c_m))\}\rangle$ *is satisfiable*

Hence, a MUST can be interpreted as a tentative way to explain infeasibility at a lower level of abstraction than the constraints one does. At this point, it is important to note that this definition for a MUST of a CSP P does not require that allowing one of the forbidden tuples of the MUST will make P become satisfiable. Indeed, it is easy to prove that an unsatisfiable CSP can exhibit an exponential number of MUSTs in the worst case and that their set-theoretic intersection can be empty. Thus, the removal of *one* forbidden tuple might not be enough to regain satisfiability. Furthermore, as the following example shows, tuples contained in a MUST might even not take part into the actual cause of the infeasibility of the CSP.

Example 3

Let $P = \langle V, C\rangle$ s.t. $V = \{v_1, v_2, v_3\}$, with $dom(v_1) = dom(v_3) = \{A\}$, $dom(v_2) = \{A, B\}$, and $C = \{c_1 = (\{v_1, v_2\}, \{(A, B)\}), c_2 = (\{v_2, v_3\}, \{(A, A),\ (B, A)\})\}$. In Figure 2, both the graph and the micro-structure of this CSP are given. Clearly, P is unsatisfiable and exhibits only one MUC, made of the c_2 constraint, since this one prevents any assignment from being valid between v_2 and v_3. On the contrary, considering a lower level of abstraction, we see that P exhibits two MUSTs, namely:

- $P_{M_1} = \langle V, \{(\{v_1, v_2\}, \{(A, B)\}), \{(\{v_2, v_3\}, \{(A, A)\})\}\}\rangle$
- $P_{M_2} = \langle V, \{(\{v_2, v_3\}, \{(A, A),\ (B, A)\})\}\rangle$

P_{M_1} is a MUST that contains tuples from both constraints. It does not correspond to any MUC of P. P_{M_2} is a MUST that is also a MUC of P. Moreover, P_{M_1} contains the only forbidden tuple linking v_1 and v_2, which does not participate to the unsatisfiability of P.

Although these results might sound negative, in the next section it is shown that MUSTs form an adequate concept to explain unsatisfiability at the tuples level, provided that MUSTs are considered within MUCs.

4 MUSTs Within MUCs

It is well-known that any unsatisfiable CSP exhibits at least one MUC (which can be the CSP itself). Unsurprisingly, any unsatisfiable CSP also exhibits at least one MUST. Indeed, MUCs are unsatisfiable CSPs, which ensures that at least one MUST can be extracted from any MUC of a CSP.

Proposition 1. *At least one MUST can be extracted from any unsatisfiable CSP.*

Proof. *Let $P = \langle V, C\rangle$ be an unsatisfiable CSP. Assume that P does not contain any MUST. Thus, P itself is not a MUST: hence there exists a forbidden tuple of P s.t. allowing it gives rise to another unsatisfiable CSP containing no MUST, namely $\exists c \in C, \exists t \in T(c)$ such that $P^1 = \langle V, (C\backslash c) \cup (Var(c),\ T(c)\backslash t)\rangle$ is also unsatisfiable, and does not exhibit any MUST. Iterating this reasoning, it is easily proved by induction that this would lead to the existence of a CSP $P' = \langle V, \emptyset\rangle$ that should be unsatisfiable, whereas such a CSP is clearly satisfiable.* □

Moreover, stronger relations link MUCs and MUSTs, as shown by the following proposition.

Proposition 2. *Let P be a MUC that contains m constraints. There exists at least m tuples s.t. allowing any one of them makes P regain satisfiability. These tuples belong to all MUSTs of P.*

Proof. *Since $P = \langle V, C\rangle$ is a MUC, whenever any of its m constraints c_i is dropped, the resulting CSP is satisfiable. Let A be an assignment that satisfies $P' = \langle V, C\backslash c_i\rangle$. c_i is violated by A: indeed, in the opposite case, P would be satisfiable and would not be a MUC. The projection of A on $Var(c_i)$ is included in $T(c_i)$, according to definition 5. Thus, removing this forbidden tuple is sufficient to make P feasible. The same argument can be used for any of the m constraints of P. Thus, there exists at least m tuples such that omitting one of them makes the MUC regain feasibility. Clearly, these tuples necessarily belong to all the sources of unsatisfiability, and consequently to every MUST of the MUC. Hence, the set-theoretic intersection of all MUSTs of P contains at least m tuples.* □

In this respect, some tuples allow the unsatisfiability of the MUC to be "broken", simply by allowing any of them. These tuples necessarily belong to all sources of unsatisfiability of the MUC, and consequently to every MUST of the MUC. Accordingly, it is thus not possible to discover two MUSTs of a MUC with an empty set-theoretic intersection. Tuples belonging to every MUST of a MUC P will be called *shared tuples* of P.

Definition 8. *Let P be a MUC. The shared tuples of P are the forbidden tuples belonging to all MUSTs of P.*

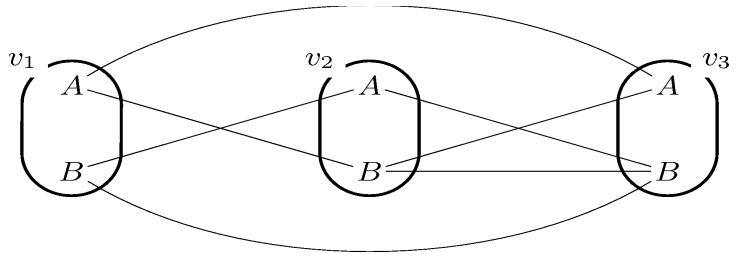


Fig. 3. Micro-structure of Example 4

Allowing any shared tuple allows the corresponding MUC to be broken. Moreover, computing a MUST of a MUC P delivers a super-set of the set of shared tuples.

Example 4. Let a CSP $P = \langle \{v_1, v_2, v_3\}, \{c_1, c_2, c_3\} \rangle$ s.t.

1. $\forall i \in \{1, 2, 3\}, dom(v_i) = \{A, B\}$
2. $c_1 = (\{v_1, v_2\}, \{(A, B), (B, A)\})$
3. $c_2 = (\{v_2, v_3\}, \{(A, B), (B, A), (B, B)\})$
4. $c_3 = (\{v_1, v_3\}, \{(A, A), (B, B)\})$

The micro-structure of P is given in Figure 3. Let us note that P is a MUC since P is unsatisfiable and dropping any of its constraints yields a satisfiable CSP. P exhibits two MUSTs; their micro-structures are given in Figure 4. By considering the set-theoretic intersection of those MUSTs, one can obtain the shared tuples of P. The shared tuples are represented using boldface edges in Figure 5, while the other tuples are represented using dotted lines. Clearly, allowing one shared tuple allows us to restore the satisfiability of both MUSTs and their corresponding MUC. However, allowing any other forbidden tuple of these MUSTs does not guarantee the satisfiability of the initial MUC to be restored. This example also shows us that the CSP formed with the shared tuples of a MUC is not necessarily unsatisfiable.

These last results plead for a two-step policy for the explanation of unsatisfiability in terms of MUCs, MUSTs and shared tuples. Indeed, looking for MUSTs in the general case does not seem the most promising approach since MUSTs can coincide with no MUC at all. On the contrary, an interesting approach would require to search for MUCs as a first step. MUCs provide explanations of unsatisfiability that are expressed in terms of a minimal number of constraints. The user can drop one such constraint to break the unsatisfiability of the MUC. Alternatively, he (she) could rather search for MUSTs corresponding to the discovered MUC. More precisely, if he (she) manages to discover shared tuples, the user would be provided with a set of forbidden tuples that is such that allowing just one such tuple is *enough* to break the unsatisfiability of the MUC. In such a way, the user would be given the ability to weaken problematic constraints instead of dropping one whole constraint. Such a policy is to be iterated until all (remaining) MUCs of the resulting CSP have been addressed.

Several algorithms have been proposed to compute one MUC. Thus, the next issue that is to be addressed is how both MUSTs and shared tuples could be computed within

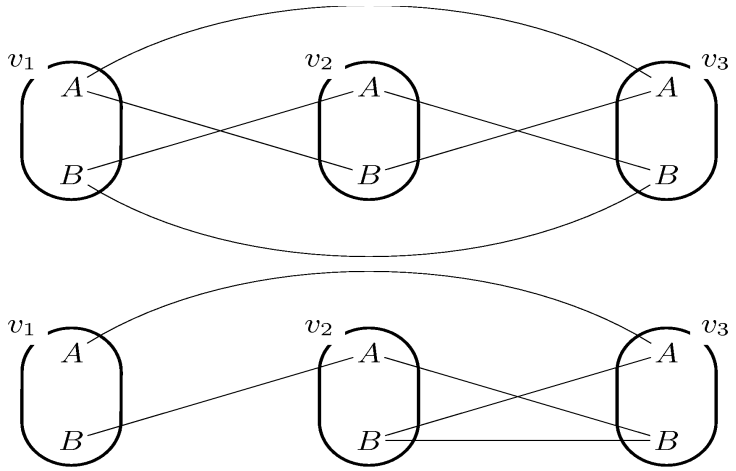


Fig. 4. Micro-structure of the two MUSTs of Example 4

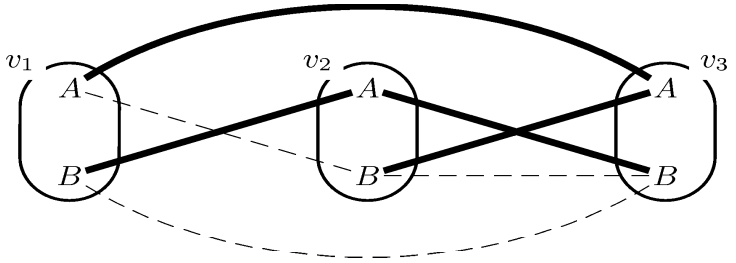


Fig. 5. Shared tuples of Example 4

a MUC. In the following, it is shown that, modulo a specific SAT encoding, shared tuples exactly coincide with so-called protected clauses [2], which play a central role in the efficiency of the currently most efficient technique to compute MUCs in the Boolean case, namely MUSes (Minimally Unsatisfiable Sub-formulas). In this respect, a Boolean translation of MUCs that allows us to benefit from the efficiency of this powerful computational technique will be provided.

5 Using OMUS to Compute MUSTs and Shared Tuples

Computing a MUST could be performed through several traditional techniques from the CSP and the operational research domains, such as the destructive, additive or dichotomic minimization procedures [4]. However, those approaches would deliver MUSTs, only. The computation of shared tuples would require either a linear number of additional step-by-step tests of satisfiability, or to consider all solutions of each relaxation obtained by removing one constraint from the computed MUC. On the contrary, an approach allowing both a MUST and shared tuples to be computed at the same time is introduced in this section.

When one MUC has been obtained trough e.g. Hemery *et al.*'s `DC(wcore)` technique [1], it is translated inside the Boolean framework in such a way that the computation of MUSTs and shared tuples of the MUC is achieved through the computation of MUSes, namely minimally unsatisfiable sets of clauses of a CNF formula. The selected translation schema is a form of *direct encoding* [5] that consists in encoding each domain value of each variable by a different Boolean variable C_{v_i}. Accordingly, the number of variables in the Boolean framework is given by the sum of the sizes of the domains of the variables of the MUC. Let $P = \langle V, C \rangle$ be a MUC, the following clauses are then created.

1. *at-Least-one clauses* ensure that at least one possible value for each variable v_i is selected in a solution
   $C_{v_1} \vee C_{v_2} \vee ... \vee C_{v_m}$ $\forall v \in V$ with $dom(v) = \{v_1, v_2, ..., v_m\}$
2. *at-Most-one clauses* ensure that at most one value is selected for each variable
   $\neg C_{v_a} \vee \neg C_{v_b}$ $\forall v \in V$ $\forall (v_a, v_b) \in dom(v) \times dom(v)$
3. *Conflict clauses* encode forbidden tuples
   $\neg C_{v_i} \vee \neg C_{v_j}$ $\forall c \in C$ $\forall (v_i, v_j) \in T(c)$

This form of encoding has been adopted because it allows a forbidden tuple to be translated into a unique clause. Moreover, the "at-Most-one clauses" are actually not added to the generated formula. Indeed, those clauses have been proved optional [6]; moreover, omitting them enables us to ensure that minimality is preserved in both frameworks. Thus, each MUS of the generated Boolean formula corresponds to a MUST of the infeasible CSP. Computing one MUST amounts to computing one MUS in this Boolean framework, provided that every *at-Least-one* clauses belong to the MUS.

More precisely, we make use of the `OMUS` technique by Grégoire *et al.* [2], which is currently one of the most efficient complete techniques to discover one MUS inside an unsatisfiable SAT instance. One key to the efficiency of this approach is the concept of *protected clauses*. Interestingly enough, protected clauses appear to encode shared tuples.

The `OMUS` technique is based on the so-called critical clause concept, which is a clause that is falsified under a given assignment of values and that is such that satisfying it by a minimal change of the assignment will always conduct at least another clause to be falsified in its turn. Roughly, the `OMUS` technique is a two-step approach. First, a superset of a MUS is computed by iterating the computation of the number of times each clause is critical during local search runs and by dropping clauses with the lowest scores. Second, a fine-tune process allows an exact MUS to be delivered. Interestingly, each time there is just one clause that is not satisfied w.r.t. some variables assignment during the local search run, it is marked and never be dropped from the formula. These clauses are called *protected*, and belong to all MUSes of the Boolean formula. According to this encoding, protected clauses coincide with shared tuples of the translated MUC. These tuples are thus delivered together with the MUST, without any computing overhead. Furthermore, the first step of this local-search-based algorithm sometimes delivers an unsatisfiable set of protected clauses, which forms a MUS, and the second step of the algorithm is avoided.

Algorithm 1. `direct_encode`

Input: a CSP: $\langle V, C\rangle$
Output: a set of clauses Σ viewed as a CNF formula

begin
  $\Sigma \longleftarrow \emptyset$;
  foreach $v_i \in V$ **do**
    $\Sigma \longleftarrow \Sigma \bigcup \{\bigvee_{\forall j \in dom(v_i)} x_{ij}\}$; /* "At-least-one" clauses */
  foreach $c \in C$ **do**
    foreach $t \in T(c)$ **do**
      $\Sigma = \Sigma \bigcup \{\bigvee_{\forall (v_i \in Var(c) \text{ and } j \in dom(v_i)) \text{ s.t. } j \text{ is the forbidden value of } v_i \text{ in } t} \neg x_{ij}\}$;
  return Σ ;
end

Algorithm 2. `mus2must`

Input: a MUS: Σ and a MUC: $\langle V, \{(Var(c_1), T(c_1)), ..., (Var(c_m), T(c_m))\}\rangle$
Output: a MUST: $\langle V, \{(Var(c_1), T'(c_1)), ..., (Var(c_m), T'(c_m))\}\rangle$

begin
  foreach $c \in C$ *s.t.* $C = \{(Var(c_1), T(c_1)), ..., (Var(c_m), T(c_m))\}$ **do**
    $T'(c) \longleftarrow \emptyset$;
    foreach $t \in T(c)$ **do**
      if $((\bigvee_{\forall (v_i \in Var(c) \text{ and } j \in dom(v_i)) \text{ s.t. } j \text{ is the forbidden value of } v_i \text{ in } t} \neg x_{ij}) \in \Sigma)$ **then**
        $T'(c) \longleftarrow T'(c) \cup \{t\}$;
  return $\langle V, \{(Var(c_1), T'(c_1)), ..., (Var(c_m), T'(c_m))\}\rangle$;
end

6 Experimental Studies

In order to assess the practical value of these techniques, an algorithm called `MUSTER` (MUST-ExtRaction) has been implemented and run on various benchmarks from the last CSP competition [7]. This software makes thus use of (a `C` variant of) Hemery *et al.*'s `DC(wcore)` technique to extract a MUC from the considered CSP. Then, the MUC is converted into a Boolean clausal formula according to the aforementioned *direct-encoding* described in Algorithm 1. A MUS is then computed thanks to Grégoire *et al.*'s `OMUS` procedure [2]. As described above, there is a one-to-one correspondence between this MUS and a MUST from the CSP; this latter one is delivered, together with detected shared tuples, using a simple translation procedure described in Algorithm 2. The `MUSTER` method is summarized in Algorithm 3.

All experiments have been conducted on a Pentium IV, 3Ghz under Linux Fedora Core 5. A significant sample of results are given on Table 1, which contains 3 main columns, namely *Instance*, *Extracted MUC* and *Extracted MUST*. The first column provides information about the considered unsatisfiable CSP: namely, the benchmark

Algorithm 3. `MUSTER`

Input: a CSP: $\langle V, C \rangle$
Output: a MUST: $\langle V, \{(Var(c'_1), T''(c'_1)), ..., (Var(c'_m), T''(c'_m))\}\rangle$

begin
  $\langle V, C' \rangle \longleftarrow$ `DC(wcore)` $(\langle V, C \rangle)$; /* $\langle V, C' \rangle$ is a MUC s.t. $C' \subseteq C$ */
  /* and $C' = \{(Var(c'_1), T(c'_1)), ..., (Var(c'_m), T(c'_m))\}$ */
  $\Sigma_{CNF} \longleftarrow$ direct_encode($\langle V, C' \rangle$) ;
  $\Sigma_{MUS} \longleftarrow$ `OMUS` (Σ_{CNF}) ; /* $\Sigma_{MUS} \subseteq \Sigma_{CNF}$ */
  $\langle V, \{(Var(c'_1), T''(c'_1)), ..., (Var(c'_m), T''(c'_m))\}\rangle \longleftarrow$ mus2must(Σ_{MUS} , $\langle V, C' \rangle$);
  /* $\forall \quad 1 \leq i \leq m \quad T''(c'_i) \subseteq T(c'_i)$ */
  return $\langle V, \{(Var(c'_1), T''(c'_1)), ..., (Var(c'_m), T''(c'_m))\}\rangle$;
end

name, the number of involved constraints and the number of forbidden tuples that the constraints represent. In the second one, the main information about the MUC computed by `DC(wcore)` is given: namely, the number of constraints of the MUC, the number of tuples that they represent, and the computing time in seconds that was spent. Finally, the third main column provides the main information about the computed MUST: namely, its number of tuples, the number of discovered shared tuples, and the computing time in seconds that was spent. A time-out was set to 3 hours of CPU time.

First, let us note that a MUST was extracted within a reasonable amount of time for most benchmarks, which represent hard to solve problems. For instance, a MUC made of 13 constraints is discovered for the `composed-75-1-2-1` CSP, which contains 624 contraints. This MUC forbids 845 tuples. Actually, this set can be reduced to just 344 tuples, which form one MUST of the benchmark. Moreover, the user is provided with a set of 104 tuples such that if one of these latter tuples is allowed, then the unsatisfiable part of the CSP represented by the MUC is fixed. Similar results were obtained for e.g. `scen11_f10`, which is an instance of the famous Radio Link Frequency Assignment Problem (RLFAP). This latter benchmark involves almost 800,000 forbidden tuples. However, only some of these them really participate to the unsatisfiability of the CSP. Indeed, a MUC that contains less than 5,000 tuples is exhibited, and this one has been reduced into a MUST made of 3,077 tuples. In this MUST, allowing one tuple among the 2,728 discovered shared ones is enough to allow the MUC to regain feasibility.

Obviously enough, due to the high-level computational complexity of the addressed problem, we cannot expect our approach to solve all problems within a reasonable amount of time. For example, although a MUST was extracted for the Queen-Knight problem `qk_8_8_5_add`, `MUSTER` was not able to deliver any of its shared tuples, although we know that these latter tuples are contained in the 10149 tuples that form the computed MUST. Let us also note that the same MUC and MUST have been discovered for both `qk_8_8_5_add` and `qk_8_8_5_mul` problems. This is easily explained: those problems result from various combinations between the 8 queens problem and 5 knights one. Since the 5 knights problem is not feasible, a same explanation of unsatisfiability can be delivered for all combinations of this problem with other CSPs.

Table 1. Extracting a MUST

Instance			**Extracted MUC**			**Extracted MUST**		
name	#con	#tuples	#con	#tuples	time (s)	#tuples	#st[1]	time (s)
composed-25-1-2-0	224	4,440	14	910	10.72	354	119	13.08
composed-25-1-2-1	224	4,440	15	975	9.09	339	59	17.47
composed-25-1-25-8	247	4,555	9	585	8.85	259	116	6.25
composed-75-1-2-1	624	10,440	13	845	66.42	344	104	10.85
composed-75-1-2-2	624	10,440	14	910	66.74	376	48	14.44
composed-75-1-25-8	647	10,555	16	1,040	59.09	461	51	23.69
composed-75-1-80-6	702	10,830	11	715	61.48	278	55	8.17
composed-75-1-80-7	702	10,830	16	1,040	379.85	420	75	17.23
composed-75-1-80-9	702	10,830	12	780	86.16	306	89	9.01
qk_10_10_5_add	55	48,640	5	47,120	19.68	24,855	0	3081.1
qk_10_10_5_mul	105	49,140	5	47,120	1.29	24,855	0	2812.99
qk_8_8_5_add	38	19,624	5	18,800	3.33	10,149	0	544.7
qk_8_8_5_mul	78	19,944	5	18,800	0.66	10,149	0	531.24
graph2_f25	2,245	145,205	43	4,498	427.36	2470	1,516	426.05
qa_3	40	800	15	583	0.32	203	152	8.32
dual_ehi-85-297-14	4,111	102,234	40	1,145	3.35	311	142	40.26
dual_ehi-85-297-15	4,133	102,433	35	1,083	4.03	310	172	25.85
dual_ehi-85-297-16	4,105	102,156	36	1,032	4.68	301	159	29.05
dual_ehi-85-297-17	4,102	102,112	43	1,239	4.83	348	172	42.21
dual_ehi-85-297-18	4,120	102,324	33	972	3.48	271	141	30.4
dual_ehi-90-315-21	4,388	108,890	37	1,120	3.16	354	129	35
dual_ehi-90-315-22	4,368	108,633	41	1,218	4.57	410	187	43.69
dual_ehi-90-315-23	4,375	108,766	29	835	2.86	251	131	12.23
dual_ehi-90-315-24	4,378	108,793	31	974	4.57	315	167	25.42
dual_ehi-90-315-25	4,398	108,974	38	1,106	3.89	375	179	30.41
scen6_w2	648	513,100	7	8,020	53.78	4,872	2,953	1107.39
scen6_w1_f2	319	274,860	21	21,146	488.64	-	-	*time out*
scen11_f10	4,103	738,719	16	4,588	164.27	3,077	2,728	438.26
scen11_f12	4,103	707,375	16	4,588	122.12	3,053	2,728	419.83

Finally, let us comment on the number of shared tuples. Proposition 2 ensures that a MUC made of m constraints contains at least m shared tuples. However, the number of shared tuples is often larger in practice. For example, a MUC made of 43 constraints is extracted from `graph2_f25`: so, at least 43 shared tuples could have been expected. Actually, 1,516 such tuples were delivered, enabling the user to just select one constraint among 43 ones, and then just select and allow one tuple among an average of 35 candidate ones, to regain feasibility.

7 Related Works

The approach introduced in this paper can be interpreted as a refinement of explanation techniques that provide users with MUCs in case of unsatisfiability. There have been

[1] #st: #shared tuples.

only a few research results about extracting MUCs from CSPs, or MUSes in the Boolean case. In this respect, the approach in this paper takes advantage of one of the most often efficient technique to compute MUCs [1] and of the currently most efficient technique compute MUSes [2], in order to deliver MUSTs and shared tuples.

In the CSP framework, there have been several other works about the identification of (minimal) conflict sets of constraints (e.g. [8]) that are recorded during the search in order to perform various forms of intelligent backtracking, like dynamic backtracking [9] [10] or conflict-based backjumping [11]. In [12] a non-intrusive method was proposed to detect them. However, there have been few research works about the problem of extracting MUCs themselves. A method to find all MUCs from a given set of constraints has been presented in [13] and in [14], which corresponds to an exhaustive exploration of a so-called CS-tree but is limited by the combinatorial blow-up in the number of subsets of constraints. Other approaches are given in [15] and in [16], where an explanation that is based on the user's preferences is extracted. Also, the `PaLM` framework [17], implemented in the constraint programming system `Choco` [18], is an explanation tool that can answer for instance the question: why is there no solution that contains the value v_i for some variable A? Moreover, in case of unsatisfiability, `PaLM` is able to provide a core, but this one is not guaranteed to be minimal. The `DC(wcore)` approach that is used in this paper appears to improve a previous method introduced in [19] to extract a MUC, that was proposed in the specific context of model-based diagnosis. It also proves more competitive than the use of the `QuickXPlain` [12] method to compute MUCs.

In the Boolean framework, the problem of extracting a MUS from an unsatisfiable CNF formula has also received much attention. In [20], Bruni has proposed an approach that approximates MUSes by means of an adaptative search guided by clauses hardness. Zhang and Malik have described in [21] a way to extract MUSes by learning nogoods involved in the derivation of the empty clause by resolution. In [22], Lynce and Marques-Silva have proposed a complete and exhaustive technique to extract one smallest MUS of a SAT instance. Together with Mneimneh, Andraus and Sakallah [23], the same authors have also proposed an algorithm that makes use of iterative max-SAT solutions to compute such smallest unsatisfiable subsets of clauses. Oh and her co-authors have presented in [24] a Davis, Putnam, Logemann and Loveland DPLL-oriented approach that is based on a marked clause concept to allow one to approximate MUSes. Let us also mention a complete approach by Liffiton and Sakallah [25], recently improved by a non-standard use of local search [26], that attempts to compute the exhaustive set of MUSes of a propositional formula.

Finally, let us note that the problem of finding an *Irreductible Infeasible Subsystem* has also been the subject of specific research efforts in mathematical programming [4][27].

8 Conclusions and Perspectives

These results open many interesting research perspectives.

An unsatisfiable CSP can exhibit several MUCs that can share non-empty set-theoretic intersections. Regaining feasibility through permitting shared tuples thus

requires the MUSTER process to be iterated until all remaining MUCs in the resulting CSP have been addressed. Clearly, the order according to which MUCs are addressed influences the tuples that are delivered to the users. In this respect, it could be fruitful to develop order-independent techniques that directly deliver sets of tuples that would make the CSP feasible if these latter tuples were allowed. In this respect, recent results by Grégoire *et al.* [28] that allows covers of MUSes to be computed, i.e. sets of MUSes that cover all basic infeasibility causes, could be exploited in that direction. The concept of shared tuples would have to be revised accordingly, and specific computational techniques would have to be devised in order to compute that.

Although it appears to be highly competitive in practice, our technique to compute shared tuples remains uncomplete since it does not guarantee that all shared tuples will be delivered. Another interesting path for future research would consist in developping complete techniques that guarantee, modulo a possible exponential blow-up, the computation of *all* shared tuples.

In this paper, basic formal results about MUSTs have been provided. Clearly, much more can be done in this respect. Variant definitions coud be provided, in particular variants that would not rely on the MUC concept to address infeasibility.

Also, several MUSTs can be inter-dependent within a CSP: studying the formal properties of their relationships is also a promising path for future research.

References

1. Hemery, F., Lecoutre, C., Saïs, L., Boussemart, F.: Extracting MUCs from constraint networks. In: ECAI'06. Proceedings of the 17th European Conference on Artificial Intelligence, pp. 113–117 (2006)
2. Grégoire, E., Mazure, B., Piette, C.: Extracting MUSes. In: ECAI'06. Proceedings of the 17th European Conference on Artificial Intelligence, pp. 387–391 (2006)
3. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates and counterfactual. Artificial Intelligence 57, 227–270 (1992)
4. Chinneck, J.: Feasibility and Viability. In: Advances in Sensitivity Analysis and Parametric Programming, ch. 14, vol. 6. Kluwer Academic Publishers, Boston (USA) (1997)
5. de Kleer, J.: A comparison of ATMS and CSP techniques. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89), pp. 290–296 (1989)
6. Walsh, T.: SAT v CSP. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 441–456. Springer, Heidelberg (2000)
7. CSPcomp: CSP competition, http://cpai.ucc.ie/06/competition.html
8. Petit, T., Bessière, C., Régin, J.: A general conflict-set based framework for partial constraint satisfaction. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, Springer, Heidelberg (2003)
9. Ginsberg, M.L.: Dynamic backtracking. Journal of Artificial Intelligence Research 1, 25–46 (1993)
10. Jussien, N., Debruyne, R., Boizumault, P.: Maintaining arc-consistency within dynamic backtracking. In: Principles and Practice of Constraint Programming, pp. 249–261 (2000)
11. Prosser, P.: Hybrid algorithms for the constraint satisfaction problems. Computational Intelligence 9(3), 268–299 (1993)
12. Junker, U.: QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. In: IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1) (2001)

13. Han, B., Lee, S.: Deriving minimal conflict sets by CS-Trees with mark set in diagnosis from first principles. IEEE Transactions on Systems, Man, and Cybernetics 29, 281–286 (1999)
14. de la Banda, M., Stuckey, P.J., Wazny, J.: Finding all minimal unsatisfiable subsets. In: Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDL'03), pp. 32–43 (2003)
15. Mauss, J., Tatar, M.M.: Computing minimal conflicts for rich constraint languages. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02), pp. 151–155 (2002)
16. Junker, U.: QuickXplain: Preferred explanations and relaxations for over-constrained problems. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04), pp. 167–172 (2004)
17. Jussien, N., Barichard, V.: The PaLM system: explanation-based constraint programming. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 118–133. Springer, Heidelberg (2000)
18. Laburthe, F., Team, T.O.P.: Choco: implementing a cp kernel. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, Springer, Heidelberg (2000), `http://www.choco-constraints.net`
19. Bakker, R.R., Dikker, F., Tempelman, F., Wognum, P.M.: Diagnosing and solving overdetermined constraint satisfaction problems. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), vol. 1, pp. 276–281. Morgan Kaufmann, San Francisco (1993)
20. Bruni, R.: Approximating minimal unsatisfiable subformulae by means of adaptive core search. Discrete Applied Mathematics 130(2), 85–100 (2003)
21. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable boolean formula. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919. Springer, Heidelberg (2004)
22. Lynce, I., Marques-Silva, J.: On computing minimum unsatisfiable cores. In: International Conference on Theory and Applications of Satisfiability Testing (2004)
23. Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques Silva, J.P., Sakallah, K.A.: A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 467–474. Springer, Heidelberg (2005)
24. Oh, Y., Mneimneh, M., Andraus, Z., Sakallah, K., Markov, I.: AMUSE: a minimally-unsatisfiable subformula extractor. In: Proceedings of the 41th Design Automation Conference (DAC 2004), pp. 518–523 (2004)
25. Liffiton, M., Sakallah, K.: On finding all minimally unsatisfiable subformulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 173–186. Springer, Heidelberg (2005)
26. Grégoire, E., Mazure, B., Piette, C.: Boosting a complete technique to find MSSes and MUSes thanks to a local search oracle. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), vol. 2, pp. 2300–2305 (2007)
27. Atlihan, M., Schrage, L.: Generalized filtrering algorithms for infeasibility analysis. Computers and Operations Research (to appear, 2007)
28. Grégoire, E., Mazure, B., Piette, C.: Local-search extraction of MUSes. Constraints Journal: Special issue on Local Search in Constraint Satisfaction 12(3) (to appear, 2007)

Tractable Cover Compilations

Article publié dans les actes de « *the Fifteenth International Joint Conference on Artificial Intelligence* », volume 1, pages 122-127, 1997.

Co-écrit avec Yacine BOUFKHAD, Éric GRÉGOIRE, Pierre MARQUIS et Lakhdar SAÏS.

Tractable Cover Compilations*

Yacine Boufkhad[1], **Éric Grégoire**[2], **Pierre Marquis**[2],
Bertrand Mazure[2], **Lakhdar Saïs**[2,3]

[1] LIP6
Université Paris 6
4, place Jussieu
F-75252 Paris Cedex 05, FRANCE
boufkhad@laforia.ibp.fr

[2] CRIL [3] IUT de Lens
Université d'Artois
Rue de l'Université, S.P. 16
F-62307 Lens Cedex, FRANCE
{gregoire,marquis,mazure,sais}@cril.univ-artois.fr

Abstract

Tractable covers are introduced as a new approach to equivalence-preserving compilation of propositional knowledge bases. First, a general framework is presented. Then, two specific cases are considered. In the first one, partial interpretations are used to shape the knowledge base into tractable formulas from several possible classes. In the second case, they are used to derive renamable Horn formulas. This last case is proved less space-consuming than prime implicants cover compilations for every knowledge base. Finally, experimental results show that the new approaches can prove efficient w.r.t. direct query answering and offer significant time and space savings w.r.t. prime implicants covers.

1 Introduction

Different approaches have been proposed to circumvent the intractability of propositional deduction. Some of them restrict the expressive power of the representation language to tractable classes, like the Horn, reverse Horn, binary, monotone, renamable Horn, q-Horn, nested clauses formulas [Dowling and Gallier, 1984; Lewis, 1978; Boros *et al.*, 1994; Knuth, 1990]. Unfortunately, such classes are not expressive enough for many applications. Contrastingly, *compilation approaches* apply to full propositional-logic knowledge bases (KBs for short). Thanks to an off-line pre-processing step, a KB Σ is compiled into a formula Σ^* so that on-line query answering can be performed tractably from Σ^*. Many approaches to compilation have been proposed so far, mainly [Reiter and De Kleer, 1987; Selman and Kautz, 1991; 1994; del Val, 1994; Dechter and Rish, 1994; Marquis, 1995; del Val, 1995; 1996; Marquis and Sadaoui, 1996; Schrag, 1996].

*This work has been supported in part by the Ganymède II project of the Contrat État/Région Nord–Pas-de-Calais.

In this paper, a new approach to equivalence-preserving compilation, called *tractable covers*, is introduced. In short, a tractable cover of Σ is a finite set $\mathcal{T}$ of tractable formulas Φ (disjunctively considered) s.t. $\Sigma \equiv \mathcal{T}$. Tractable covers of Σ are equivalence-preserving compilations of Σ: a clause c is a logical consequence of Σ iff for every Φ in $\mathcal{T}$, c is a logical consequence of Φ. Since Φ is tractable, each elementary test $\Phi \models c$ can be computed in time polynomial in $|\Phi| + |c|$. The point is to find out tractable Φs that concisely represent (i.e. cover) the largest sets of models of Σ, so that $|\mathcal{T}|$ remains limited. To some extent, the present work could then be related to other model-based approaches to knowledge representation and reasoning, like [Khardon and Roth, 1996].

First, a general framework is presented, which can take advantage of most tractable classes, simultaneously. In many respects, it generalizes the prime implicants cover technique recently used for compilation purpose [Schrag, 1996]. Then, the focus is laid on tractable covers that can be computed and intensionally represented thanks to (partial) interpretations. Two specific cases are considered. In the first one, partial interpretations are used to shape the KB into formulas from several possible classes. In the second one, they are used to derive renamable Horn formulas. The last one is proved less space-consuming than prime implicants covers [Schrag, 1996] for every KB. Since tractable covers of Σ are equivalence-preserving compilations, their size may remain exponential in $|\Sigma|$ unless $NP \subseteq P/poly$ [Selman and Kautz, 1994], which is very unlikely. However, experimental results show that the new approaches can prove efficient w.r.t. direct query answering and offer significant time and space savings w.r.t. prime implicants covers.

2 Formal Preliminaries

A literal is a propositional variable or a negated one. A clause (resp. a term) is a finite set of literals, representing their disjunction (resp. conjunction). A Horn (resp. reverse Horn) clause contains at most one literal that is

positive (resp. negative). A binary clause contains at most two literals. A KB Σ is a finite set of propositional formulas, conjunctively considered. $Var(\Sigma)$ is the set of variables occurring in Σ. Every KB can be rewritten into a conjunction of clauses (into CNF for short) while preserving equivalence. A KB Σ is said renamable Horn iff there exists a substitution σ over literals l built up from $Var(\Sigma)$, s.t. $\sigma(l) = \bar{l}$ and $\sigma(\Sigma)$ is Horn.

Interpretations and models are defined in the usual way. Let us stress that, quite unconventionally, (partial) interpretations will be represented as terms, i.e. as satisfiable sets of literals (vs. sets of variables). An implicant of a KB Σ is a partial interpretation α s.t. $\alpha \models \Sigma$. A prime implicant of Σ is an implicant π of Σ s.t. for all implicants α of Σ s.t. $\pi \models \alpha$, we have $\alpha \models \pi$. The set of all prime implicants of Σ (up to logical equivalence) is noted $PI(\Sigma)$. A prime implicant cover of Σ is any subset S of $PI(\Sigma)$ (disjunctively considered) s.t. $S \equiv \Sigma$.

3 Tractable Cover Compilations

3.1 The General Framework

First, let us make precise what classes of tractable formulas will be considered:

Definition 3.1 *A list Cs of* classes C of tractable propositional formulas *(w.r.t. cover compilation) is s.t.:*

- *There exists a polytime algorithm $TRACTABLE?$ for checking whether any formula Φ belongs to C.*
- *There exists a polytime algorithm $QUERY?$ for checking whether $\Phi \models c$ holds for any Φ in C and any clause c.*
- *For every C in Cs, for every formula Φ of C and every term p, there exists C' in Cs s.t. $(\Phi \wedge p) \in C'$.*

Since classes of tractable formulas are not finite sets in the general case, they are intensionally represented by ordered pairs of decision procedures $\langle TRACTABLE?, QUERY? \rangle$.

Interestingly, the great majority of classes of formulas tractable for SAT (the well-known propositional satisfiability decision problem) are also tractable for cover compilations. Especially, this is the case for the Horn, reverse Horn, binary, renamable Horn, q-Horn and nested clauses classes.

We are now ready to define the notion of a tractable cover of a propositional KB.

Definition 3.2 *Given a CNF-KB Σ and a finite list Cs of classes of tractable formulas w.r.t. cover compilation (represented intensionally), a* tractable cover *of Σ w.r.t. Cs is a finite set $\mathcal{T}$ of satisfiable formulas (disjunctively considered) s.t.:*

- *For every $\Phi_i \in \mathcal{T}$, there exists C in Cs s.t. Φ_i belongs to C, and*
- $\mathcal{T} \equiv \Sigma$.

In the following, we will assume that Cs contains at least the class $\{t$ s.t. t is a term$\}$. This ensures that there always exists at least one tractable cover of Σ, namely a prime implicants one.

In this paper, we focus on tractable covers that can be *intensionally represented*, using (partial) interpretations. The corresponding explicit covers can be generated on-line from the intensional ones in polynomial time.

3.2 Carver-Based Tractable Covers

A first way to derive covers consists in *simplifying* the KB using partial interpretations. For any partial interpretation $p = \{l_1, \ldots, l_n\}$, the KB Σ simplified w.r.t. p is the set $\Sigma_p = (((\Sigma_{l_1})_{l_2}) \ldots)_{l_n}$, where $\Sigma_l = \{c \setminus \{\neg l\}$ s.t. $c \in \Sigma$ and $c \cap \{l\} = \emptyset\}$.

Definition 3.3 *Given a CNF-KB Σ and a finite list Cs of classes of tractable formulas (represented intensionally):*

- *A* carver *of Σ w.r.t. Cs is a partial interpretation p s.t. there exists C in Cs s.t. Σ_p belongs to C and Σ_p is satisfiable.*
- *A* carver-based tractable cover *of Σ w.r.t. Cs is a set PC of carvers of Σ w.r.t. Cs (disjunctively considered) s.t. $(PC \wedge \Sigma) \equiv \Sigma$.*
- *A* carver-based cover compilation *of Σ is a pair $\mathcal{C} = \langle \Sigma, PC \rangle$, where PC is a carver-based tractable cover of Σ w.r.t. Cs.*

Interestingly, the space required to store a carver p is always lower or equal to the space needed by the corresponding tractable formula $\Phi = p \wedge \Sigma_p$ (with a $O(|\Sigma|)$ factor in the worst case). At the on-line query answering stage, the price to be paid is a $O(|\Sigma \wedge p|)$ time complexity overhead per carver p but this does not question the tractability of query answering.

As an example, let $\Sigma = \{\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4, x_1 \vee x_2 \vee x_4\}$ and let Cs contain the Horn and the binary classes. The set $PC = \{\{x_1\}, \{\bar{x}_1\}\}$ is a carver-based tractable cover of Σ w.r.t. Cs since $\Sigma_{\{x_1\}} = \{\bar{x}_2 \vee \bar{x}_3 \vee x_4\}$ is Horn and $\Sigma_{\{\bar{x}_1\}} = \{x_2 \vee x_4\}$ is binary. If Cs reduces to the renamable Horn class, $\{\emptyset\}$ is a carver-based tractable cover of Σ w.r.t. Cs since Σ belongs to the renamable Horn class.

Clearly enough, carver-based cover compilations are equivalence-preserving compilations:

Proposition 3.1 *Let Σ be a CNF-KB, $\mathcal{C} = \langle \Sigma, PC \rangle$ be a carver-based cover compilation of Σ and let c be a clause. $\Sigma \models c$ iff for every p in PC, $((p \models c)$ or $(\Sigma_p \models c))$, which can be decided in time polynomial in $|\mathcal{C}| + |c|$.*

Although the class(es) to which Σ_p belongs can be polynomially recognized on-line for each carver p of PC, in practice, they are determined once only at the off-line compiling stage. Accordingly, each carver p found during the compiling process is indexed with a reference $label(p)$ to the concerned class in Cs. A significant amount of time in on-line query answering can be saved via such indexing.

Interestingly, carver-based cover compilations can lead to exponential space savings w.r.t. prime implicants

cover compilations for some KBs. An extreme case consists of tractable KBs Σ that remain invariant under carver-based cover compilation but are such that every prime implicant cover is exponential in the size of Σ.

3.3 Hyper-Implicant Covers

Formulas in covers must be tractable and exhibit as many models of Σ as possible. Interestingly, several classes C of tractable formulas admit model-theoretic characterizations and features that can be exploited. For the binary, Horn, reverse Horn, renamable Horn classes, among others, class membership is equivalent to the existence of some specific interpretation(s). Formally, a CNF-KB Σ is renamable Horn [Lewis, 1978] iff there exists an interpretation p over $Var(\Sigma)$ s.t. for every clause c of Σ, at most one literal of c does not belong to p. Interestingly, the renamable Horn class includes both the Horn, reverse Horn, and satisfiable binary KBs as proper subsets.

As the semantical characterizations above indicate it, literals that belong both to p and to Σ play a central role that we can take advantage of. In order to derive satisfiable renamable Horn formulas, every clause c of Σ is shortened using a model $p = m$ of Σ: only the literals occurring in m are kept, plus an additional literal of c (when possible). In this way, every shortened clause is such that every literal of m (except possibly one) belongs to it. Accordingly, the resulting set of clauses, called *hyper-implicant* of Σ, is satisfiable and renamable Horn.

Formally, let m be a model of Σ. For every clause c in Σ, let c^m be the clause consisting of the literals common to c and m. Let $P(\Sigma, m)$ denote the set of clauses of Σ s.t. for every clause c in $P(\Sigma, m)$, c and c^m are identical. Let $N(\Sigma, m)$ be $\Sigma \setminus P(\Sigma, m)$. For every clause c in $N(\Sigma, m)$, let l_c^m denote a literal of c s.t. $\overline{l_c^m}$ belongs to m. Obviously, several candidates l_c^m may exist in the general case.

Definition 3.4 *Given a CNF-KB Σ:*

- *A* hyper-implicant *of Σ (w.r.t. to a model) m of Σ is a formula* $\Sigma^m = (\bigwedge_{c \in P(\Sigma,m)} c) \wedge (\bigwedge_{c \in N(\Sigma,m)} (c^m \vee l_c^m))$.
- *A* hyper-implicant cover $\mathcal{H}$ *of Σ is a set of hyper-implicants of Σ (disjunctively considered) s.t. $\mathcal{H} \equiv \Sigma$.*

Importantly, the size of any hyper-implicant Σ^m of Σ is strictly lower than the size of Σ, except when Σ is renamable Horn.

Hyper-implicant covers are equivalence-preserving compilations:

Proposition 3.2 *Let Σ be a CNF-KB, $\mathcal{H}$ be a hyper-implicant cover of Σ and c be a clause. $\Sigma \models c$ iff for every hyper-implicant Σ^m of $\mathcal{H}$, we have $\Sigma^m \models c$. This can be decided in time linear in $|\mathcal{H}| + |c|$.*

Assuming that the clauses in $\Sigma = \{c_1, \ldots, c_n\}$ are totally ordered, hyper-implicant covers can be represented intensionally by Σ and sets of pairs $\langle m, [l_{c_1}^m, \ldots, l_{c_n}^m] \rangle$ where m is a model of Σ and $l_{c_i}^m$ $(i \in 1..n)$ is $false$ if $c_i \in P(\Sigma, m)$ and is the literal of $c_i \setminus m$ which is kept, otherwise. Each Σ^m can be derived on-line from Σ and $\langle m, [l_{c_1}^m, \ldots, l_{c_n}^m] \rangle$ in linear time.

As an example, let $\Sigma = \{x_1 \vee x_3 \vee x_4, x_1 \vee \bar{x}_2 \vee x_3, x_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4\}$. The hyper implicant cover represented by Σ and the two pairs $\langle \{x_1, \bar{x}_2, \bar{x}_3, x_4\}, [x_3, x_3, x_2, \bar{x}_1, \bar{x}_1] \rangle$ and $\langle \{\bar{x}_1, x_2, x_3, \bar{x}_4\}, [x_1, \bar{x}_2, \bar{x}_3, \bar{x}_2, x_4] \rangle$, contains two hyper-implicants: $\{x_1 \vee x_4 \vee x_3, x_1 \vee \bar{x}_2 \vee x_3, x_1 \vee \bar{x}_3 \vee x_2, \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_1, \bar{x}_3 \vee x_4 \vee \bar{x}_1\}$ and $\{x_3 \vee x_1, x_3 \vee \bar{x}_2, x_2 \vee \bar{x}_4 \vee \bar{x}_3, \bar{x}_1 \vee \bar{x}_4 \vee \bar{x}_2, \bar{x}_1 \vee x_2 \vee x_4\}$.

Intuitively, any hyper-implicant Σ^m of Σ is a concise representation of some prime implicants of Σ. To be more specific, the prime implicants of Σ which are entailed by m are prime implicants of Σ^m.

Proposition 3.3 *For every model m of Σ, let $PI_m(\Sigma)$ be the set of prime implicants of Σ s.t. for every π in $PI_m(\Sigma)$, we have $m \models \pi$. Then, $PI_m(\Sigma) \subseteq PI(\Sigma^m)$.*

Consequently, Σ^m covers every model of Σ covered by a prime implicant of Σ entailed by m.

Hyper-implicants covers are economical representations w.r.t. prime implicants ones. Not withstanding the fact that hyper-implicants of Σ are smaller than Σ, the number of hyper-implicants in a cover is lower than the number of implicants in a cover:

Proposition 3.4 *For every prime implicant cover of Σ containing t prime implicants there exists a hyper-implicant cover of Σ containing at most $\lfloor \frac{t+1}{2} \rfloor$ hyper-implicants.*

In the example above, the hyper-implicant cover contains 2 formulas, while the smallest prime implicant cover of Σ consists of 6 prime implicants. Actually, experiments show significant savings w.r.t. the sizes of the prime implicants covers for most KBs (cf. Section 5).

4 Computing Compilations

(Partial) interpretations giving rise to intensionally-represented tractable covers are computed using systematic search, thanks to a Davis/Putnam-like procedure `DPTC`. This procedure is closely related to Schrag's `DPPI` algorithm [Schrag, 1996]. `Cs` is empty for the hyper-implicant case.

```
DPTC(Σ,Cs):
1   PC ← ∅;
2   DP*(Σ,Cs,∅);
3   Return(⟨Σ,PC⟩).

DP*(Σ,Cs,p):
1   If not PRUNING(Σ,p) then
2      UNIT_PROPAGATE(Σ,p);
3      If ∅ ∈ Σ then return;
4      If (p ⊨ Σ)
```

```
5          then PROCESS_IMPLICANT(p,Σ,Cs)
6               return;
7       l← CHOOSE_BEST_LITERAL(Σ);
8       DP*(Σ,Cs,p∪{l});
9       DP*(Σ,Cs,p∪{¬l}).
```

The CHOOSE_BEST_LITERAL branching rule and UNIT_PROPAGATE procedures are standard Davis/Putnam features. In our experiments, the branching rule by [Dubois *et al.*, 1996] is used. The main role of DP* is to find implicants of Σ in the whole search tree. PRUNING and PROCESS_IMPLICANT depend on the considered approach. From the found implicants, (partial) interpretations (carvers and implicit representations of hyper-implicants) are derived thanks to PROCESS_IMPLICANT; they are collected into the global variable PC.

4.1 The Carver-Based Case

```
PRUNING_C(Σ,new_p):
1   If there exists p ∈ PC s.t. (new_p ⊨ p)
2      then return(true) else return(false).

PROCESS_IMPLICANT_C(new_p,Σ,Cs):
1   Carver ← DERIVE_C(new_p,Σ,Cs);
2   For every p ∈ PC do
3       If (p ⊨ Carver) then remove p from PC;
4   Put Carver in PC.

DERIVE_C(p,Σ,Cs):
1   Prime ← ONE_PRIME(p,Σ);
2   For every literal l ∈ Prime do
3    For every ⟨TRACTABLE?,QUERY?⟩ in Cs do
4      If TRACTABLE?(Σ_{Prime\{l}})
         then Return(DERIVE_C(Prime\{l},Σ,Cs));
5   Return(Prime).
```

Whenever an implicant p of Σ is found, a prime implicant Prime of Σ s.t. $p \models$ Prime is extracted from p, thanks to the ONE_PRIME procedure. ONE_PRIME considers every literal l of p successively, check whether l is necessary (i.e. if there exists a clause c of Σ s.t. $c \cap p = \{l\}$), and remove l from p if l is not necessary (see [Castell and Cayrol, 1996][Schrag, 1996] for details). These prime implicants are then tentatively simplified; all the literals of each Prime are considered successively; for every literal l of Prime, $\Sigma_{Prime\setminus\{l\}}$ is checked for tractability using the recognition procedures given in Cs. If $\Sigma_{Prime\setminus\{l\}}$ is found tractable, l is ruled out from Prime and is kept otherwise. Once all the literals of Prime have been considered, the resulting simplified Prime (indexed by the label of the corresponding class in Cs) is checked against the set PC of carvers collected so far. Only maximal carvers w.r.t. $\models$ are kept in PC. Interestingly, the set PC of carvers stored during the traversal of the DP search tree is used to prune this tree, thanks to the PRUNING$_C$ procedure; indeed, whenever a candidate (partial) interpretation new_p is elected, it can be immediately removed if new_p entails one of the current carvers.

Clearly enough, both the literal ordering and the recognition procedure ordering used in DERIVE$_C$ can greatly influence the cover generated in this way.

4.2 The Hyper-Implicant Case

```
PRUNING_H(Σ,new_p):
1   If there exists p ∈ PC s.t. (Σ^{new_p} ⊨ Σ^p)
2      then return(true) else return(false).

PROCESS_IMPLICANT_H(new_p,Σ,_):
1   Model ← DERIVE_H(new_p,Σ);
2   For every p ∈ PC do
3       If (Σ^{Model} ⊨ Σ^p) then remove p from PC;
4   Put Model in PC.

DERIVE_H(p,Σ):
1   Model ← p;
2   For every variable v ∈ Var(Σ) do
3       If ({v} ∩ p = ∅) and ({¬v} ∩ p = ∅)
4          then put v with its most frequent
                sign in Σ to Model;
5   Return(Model).
```

Each time an implicant p of Σ is found, a model Model of Σ s.t. Model $\models p$ is derived from p. The variables that do not occur in p are added with the sign occurring the most frequently in Σ. When Σ^{Model} is computed for the first time, a list $[l^m_{c_1}, \ldots, l^m_{c_n}]$ is attached to $Model$. The DERIVE$_H$ procedure is both simpler and more efficient than DERIVE$_C$ (roughly, it is not more time-consuming than the prime implicant extraction achieved by ONE_PRIME). The tractable formulas Σ^p corresponding to the models p collected in PC are used to prune the search tree (PRUNING$_H$). Only the ps giving rise to the logically weakests Σ^p are kept in PC. Since the renamable Horn formulas Σ^{Model} and Σ^p stem from the same KB Σ, checking whether ($\Sigma^{Model} \models \Sigma^p$) can be performed in a very efficient way.

Both the literals chosen to extend the found implicants to models p of Σ, and the literals l^m_c selected in clauses of Σ^p may have a significant impact on the hyper-implicants Σ^p that are generated.

5 Experimental Results

In contrast to SAT, only few benchmarks for knowledge compilation can be found in the literature (with the well-developed experimental framework of [Schrag, 1996] as an exception). Actually, no comprehensive analysis of what should be the nature of meaningful benchmarks for evaluating compilation approaches has ever been conducted. Clearly, benchmarks must be hard for query answering since the goal of knowledge compilation is to overcome its intractability. However, in contrast to [Schrag, 1996], we do not focus on hard SAT instances,

problems	#var	#cla	$\alpha_{\mathcal{P}}$ $\alpha_{\mathcal{C}}$ $\alpha_{\mathcal{H}}$	$\beta_{\mathcal{P}}$ $\beta_{\mathcal{C}}$ $\beta_{\mathcal{H}}$	$\lvert\mathcal{P}\rvert$ $\lvert\mathcal{C}\rvert$ $\lvert\mathcal{H}\rvert$
adder	21	50	5.08 2.09 0.20	— — 78.75	5376 1044 305
history-ex	21	17	2.00 0.75 0.14	— 1000.00 11.66	172 234 77
two-pipes	15	54	0.66 0.60 0.12	125.00 125.00 20.00	255 234 192
three-pipes	21	82	0.47 0.42 0.27	45.45 <1 17.50	441 373 284
four-pipes	27	110	0.36 0.25 0.20	23.80 18.51 29.16	675 528 380
regulator	21	106	0.51 0.29 0.17	26.31 20.83 29.99	861 530 422
selenoid	11	19	2.16 1.40 0.20	— — 17.49	297 115 94
valve	13	50	0.85 0.66 0.16	<1 <1 21	312 297 246

Table 1: Experimental results.

only. Hard SAT instances (with respect to current algorithms) should be considered hard for query answering since at least one query (namely, the empty clause) is difficult. However, easy SAT instances can exhibit hard queries that differ from the empty clause.

Accordingly, we tested the tractable covers and the prime implicants approaches w.r.t. many KBs, including "standard" structured problems, taken from [Forbus and De Kleer, 1993], and random k-SAT problems [Dubois *et al.*, 1996], varying the $\#cla(use)/\#var(iable)$ ratio from the easy to the hard regions. Each KB Σ has been compiled using the 3 techniques. Then, 500 queries have been considered. In order to check the usefulness of the compilation process, we also answered these queries from Σ, using a direct, uncompiled, Davis/Putnam-based approach [Dubois *et al.*, 1996]. For each problem Σ and each compilation technique, the ratios $\alpha = Q_C/Q_U$ and $\beta = C/(Q_U - Q_C)$ have been computed. Q_C (resp. Q_U) is the time needed by the compiled (resp. uncompiled) approach to answer all the queries, and C is the compilation time. α (resp. β) tells us how much query time improvement we get from compilation (resp. how many queries are required to amortize the cost of compilation).

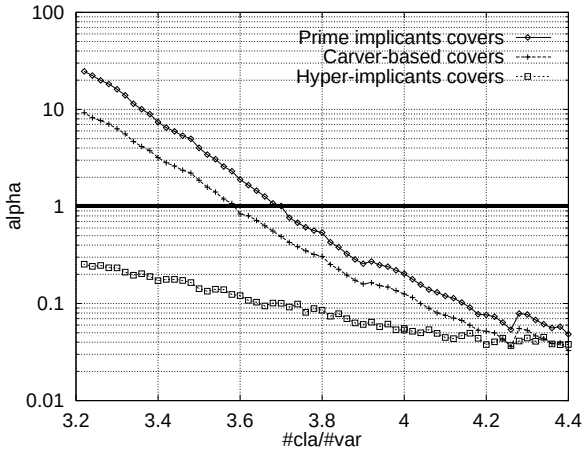


Figure 1: Ratios α.

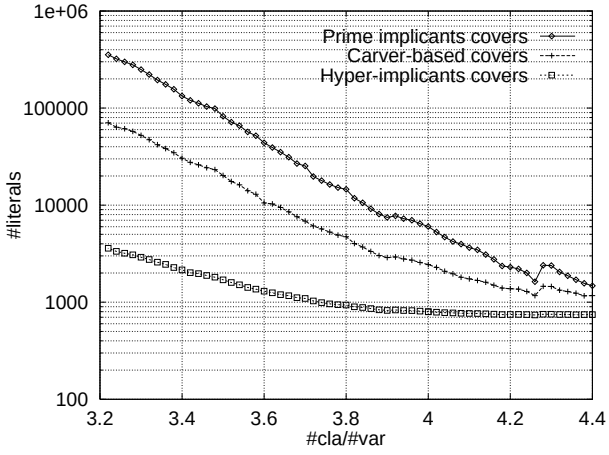


Figure 2: Sizes of the compilations.

Table 1 reports some results of our extensive experiments. For each problem [Forbus and De Kleer, 1993], it lists results for the prime implicants, carvers, and hyper-implicants covers, successively. Especially, it gives the ratios α and β and the size (in literals) of the corresponding cover. The size of any tractable cover compilation is the size of Σ plus the size of the set of (partial) interpretations used as an implicit representation. For the carver-based approach, only the Horn, reverse Horn and binary classes have been considered. For the hyper-implicant approach, simplification of the cover (i.e. lines 2 to 4 of the `PROCESS_IMPLICANT`$_{\mathcal{H}}$ procedure) has not been implemented.

Results obtained on 50 variables random 3-SAT problems, where the ratio $\#cla/\#var$ varies from 3.2 to 4.4, are reported on the two next figures. 50 problems have

been considered per point and the corresponding scores averaged. Figure 1 (resp. Figure 2) gives aggregate values of ratios α (resp. sizes in literals) obtained for each compilation technique. $\alpha = 1$ separates the region for which compilation is useful from the region for which it is not.

At the light of our experiments, tractable covers prove better than prime implicants covers, both for structured and random k-SAT problems. Significant time savings w.r.t. query answering and significant space savings are obtained. Especially, tractable covers prove useful for many KBs for which prime implicants covers are too large to offer improvements w.r.t. query answering. More, the tractable cover approach allows the compilation of KBs which have so huge prime implicants covers $\mathcal{P}$ that $\mathcal{P}$ cannot be computed and stored. This coheres with the theoretical results reported in [Boufkhad and Dubois, 1996], showing that the average number of prime implicants of k-SAT formulas is exponential in their number of variables.

6 Conclusion

Both theoretical and experimental results show the tractable cover approach promising and encourage us to extend it in several directions. A first issue for further research is how to determine efficiently the best suited classes of tractable formulas for a given KB Σ. On the experimental side, an extensive evaluation of the carver-based technique equipped with more expressive tractable classes must be done. Extending the hyper-implicant approach to other tractable classes, especially the q-Horn one [Boros *et al.*, 1994], is another interesting perspective. Finally, fragments of tractable covers of Σ can serve as approximate compilations (lower bounds) of Σ in the sense of [Selman and Kautz, 1991; 1994; del Val, 1995; 1996]. Since the tractable cover approach allows disjunctions of tractable formulas from several classes, better approximations could be obtained.

References

[Boros *et al.*, 1994] E. Boros, P.L. Hammer, and X. Sun. Recognition of q-horn formulae in linear time. *Discrete Applied Mathematics*, 55(1):1–13, 1994.

[Boufkhad and Dubois, 1996] Y. Boufkhad and O. Dubois. Length of prime implicants and number of solutions of random r-cnf formulas. *(submitted)*, 1996.

[Castell and Cayrol, 1996] T. Castell and M. Cayrol. Computation of prime implicates and prime implicants by the davis and putnam procedure. In *Proc. ECAI'96 Workshop on Advances in Propositional Deduction*, pages 61–64, Budapest, 1996.

[Dechter and Rish, 1994] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Proc. KR'94*, pages 134–145, Bonn, 1994.

[del Val, 1994] A. del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In *Proc. KR'94*, pages 551–561, 1994.

[del Val, 1995] A. del Val. An analysis of approximate knowledge compilation. In *Proc. IJCAI'95*, pages 830–836, Montreal, 1995.

[del Val, 1996] A. del Val. Approximate knowledge compilation: The first-order case. In *Proc. AAAI'96*, pages 498–503, Portland (OR), 1996.

[Dowling and Gallier, 1984] W. Dowling and J. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[Dubois *et al.*, 1996] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. Sat vs. unsat. In 2^{nd} *DIMACS Implementation Challenge*, volume 26 of *DIMACS Series*, pages 415–436. American Mathematical Society, 1996.

[Forbus and De Kleer, 1993] K.D. Forbus and J. De - Kleer. *Building Problem Solvers*. MIT Press, 1993.

[Khardon and Roth, 1996] R. Khardon and D. Roth. Reasoning with models. *Artificial Intelligence*, 87:187–213, 1996.

[Knuth, 1990] D.E. Knuth. Nested satisfiability. *Acta Informatica*, 28:1–6, 1990.

[Lewis, 1978] H.R. Lewis. Renaming a set of clauses as a horn set. *JACM*, 25:134–135, 1978.

[Marquis and Sadaoui, 1996] P. Marquis and S. Sadaoui. A new algorithm for computing theory prime implicates compilations. In *Proc. AAAI'96*, pages 504–509, Portland (OR), 1996.

[Marquis, 1995] P. Marquis. Knowledge compilation using theory prime implicates. In *Proc. IJCAI'95*, pages 837–843, Montreal, 1995.

[Reiter and De Kleer, 1987] R. Reiter and J. De Kleer. Foundations of assumption-based truth maintenance systems: Preliminary report. In *Proc. AAAI'87*, pages 183–188, Seattle (WA), 1987.

[Schrag, 1996] R. Schrag. Compilation for critically constrained knowledge bases. In *Proc. AAAI'96*, pages 510–515, Portland (OR), 1996.

[Selman and Kautz, 1991] B. Selman and H. Kautz. Knowledge compilation using horn approximations. In *Proc. AAAI'91*, pages 904–909, 1991.

[Selman and Kautz, 1994] B. Selman and H. Kautz. Knowledge compilation and theory approximation. *JACM*, 43(2):193–224, 1994.

Index

Symboles

A

B

C

Bibliographie

[Ake78] Sheldon B. AKERS. « Binary Decision Diagrams ». *IEEE Transactions on Computers*, 27(6) :509–516, 1978. 14

[ALMS09a] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Integrating Conflict Driven Clause Learning to Local Search ». Dans *Proceedings of International Workshop on Local Search Techniques in Constraint Satisfaction (affiliated to CP) (LSCS'09)*, Lisbon, Portugal, septembre 2009. Electronic proceedings. 26, 27, 87

[ALMS09b] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Learning in local search ». Dans *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI'09)*, page to appear, Newark, New Jersey, USA, novembre 2009. IEEE Computer Press. 24, 52

[AMS04] Gilles AUDEMARD, Bertrand MAZURE, et Lakhdar SAÏS. « Dealing with Symmetries in Quantified Boolean Formulas ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 257–262, 2004. 49

[Ans05] David ANSART. « *Utilisation et extensions de l'algorithmique pour SAT pour la résolution de différents problèmes d'intelligence artificielle* ». Thèse doctorat, Université d'Artois, Faculté des Sciences Jean Perrin, Lens, France, février 2005. 40

[ARMS02] Fadi A. ALOUL, Arathi RAMANI, Igor L. MARKOV, et Karem A. SAKALLAH. « Solving difficult SAT instances in the presence of symmetry ». Dans *Proceedings of the 39th annual Design Automation Conference (DAC'02)*, pages 731–736, New Orleans, Louisiana, USA, 2002. ACM, New York, NY, USA. 49

[AS07a] Gilles AUDEMARD et Lakhdar SAÏS. « Circuit Based Encoding of CNF Formula ». Dans *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 de *Lecture Notes in Computer Science*, pages 16–21, Lisbon, Portugal, 2007. Springer. 53

[AS07b] Gilles AUDEMARD et Laurent SIMON. « Gunsat : A greedy local search algorithm for unsatisfiability ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2256–2261, Hyderabad, India, janvier 6-16 2007. 20, 21, 22

[BCD+93] Salem BENFERHAT, Claudette CAYROL, Didier DUBOIS, Jérome LANG, et Henri PRADE. « Inconsistency management and prioritized syntax-based entailment ». Dans *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 640–645, Chambéry, France, September 1993. Morgan Kaufmann. 47

[BCHS90] Endre BOROS, Yves CRAMA, Peter L. HAMMER, et Michael E. SAKS. « A complexity index of satisfiability problems ». *Annals of Mathematics and Artificial Intelligence*, 1 :21–32, 1990. 31

[BDTW93] R. R. BAKKER, F. DIKKER, F. TEMPELMAN, et P. M. WOGNUM. « Diagnosing and Solving Over-Determined Constraint Satisfaction Problems ». Dans *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, volume 1, pages 276–281, Chambéry, France, September 1993. Morgan Kaufmann. 44

[Ben05] Marco BENEDDETTI. « sKizzo : A Suite to Evaluate and Certify QBFs ». Dans *Proceedings of the 20th International Conference on Automated Deduction (CADE20)*, volume 3632 de *Lecture Notes in Computer Science*, pages 369–376, Tallinn, Estonia, juillet 22-27 2005. Springer Berlin / Heidelberg. 48

[BGM+97] Yacine BOUFKHAD, Éric GRÉGOIRE, Pierre MARQUIS, Bertrand MAZURE, et Lakhdar SAÏS. « Tractable Cover Compilations ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 122–127, Nagoya, Japan, août 23-29 1997. ..... 48, 87

[BGS99] Laure BRISOUX, Éric GRÉGOIRE, et Lakhdar SAÏS. « Improving Backtrack Search for SAT by Means of Redundancy ». Dans *ISMIS '99 : Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, pages 301–309, London, UK, 1999. Springer-Verlag. 16

[BHG09] Adrian BALINT, Michael HENN, et Olivier GABLESKE. « A Novel Approach to Combine a SLS- and a DPLL-Solver for the Satisfiability Problem ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 de *Lecture Notes in Computer Science*, pages 248–297, Swansea, Wales, United Kingdom, 2009. Springer. 21, 22, 27

[BHLS04] Frédéric BOUSSEMART, Frédéric HEMERY, Christophe LECOUTRE, et Lakhdar SAIS. « Boosting systematic search by weighting constraints ». Dans *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI'99)*, pages 482–486, Valence, Spain, août 2004. 44

[BHvMW09] Armin BIERE, Marijn J. H. HEULE, Hans van MAAREN, et Toby WALSH, éditeurs. *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications*. IOS Press, février 2009. 6

[Bie05] Armin BIERE. « Resolve and Expand ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 de *Lecture Notes in Computer Science*, pages 59–70, Vancouver (BC) Canada, mai 10-13 2004. Revised selected papers published in 2005. Sringer. 48

[BJL86] Charles E. BLAIR, Robert G. JEROSLOW, et James K. LOWE. « Some results and experiments in programming techniques for propositional logic ». *Computures and Operations Research*, 13(5) :633–645, 1986. 13

[BJS97] Roberto J. BAYARDO JR. et Robert C. SCHRAG. « Using CSP Look-Back Techniques to Solve Real-World SAT Instances ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence (Rhode Island, USA), juillet 1997. 18

[BKS04] Paul BEAME, Henry A. KAUTZ, et Ashish SABHARWAL. « Towards Understanding and Harnessing the Potential of Clause Learning ». *Journal of Artificial Intelligence Research*, 22 :319–351, 2004. 18

[Boo54] George BOOLE. *Les lois de la pensée*. Mathesis, 1854. 13

[BP92] Hachemi BENNACEUR et Gérard PLATEAU. « FAST : une méthode de résolution linaire de satisfaction de contraintes ». *Techniques et Sciences Informatique*, 11(3) :33–57, 1992. 13

[BP93] Hachemi BENNACEUR et Gérard PLATEAU. « An exact algorithm for the constraint satisfaction problem : application to logical inference ». *Information Processing Letters*, 48(3) :151–158, 1993. 13

[BR00] Yacine BOUFKHAD et Olivier ROUSSEL. « Redundancy in Random SAT Formulas ». Dans *Proceedings of the Seventeenth American National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000. . . . . 27, 30

[Bra01] Ronen I. BRAFMAN. « A simplifier for Propositional Formulas with Many Binary clauses ». Dans *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, Seattle, Washington, USA, août 4-10 2001. 34

[Bru03] Renato BRUNI. « Approximating minimal unsatisfiable subformulae by means of adaptive core search ». *Discrete Applied Mathematics*, 130(2) :85–100, 2003. 40

[Bry92] Randal E. BRYANT. « Symbolic Boolean manipulation with ordered binary-decision diagrams ». *ACM Computing Surveys*, 24(3) :293–318, 1992. 14

[BS94] Belaïd BENHAMOU et Lakhdar SAÏS. « Tractability Through Symmetries in Propositional Calculus ». *Journal of Automated Reasoning*, 12 :89–102, 1994. 27

[BS05] James BAILEY et Peter J. STUCKEY. « Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization ». Dans *Proceedings of the International Symposium on Practical Aspects of Declarative Languages (PADL'05)*, pages 174–186, Long Beach (États-Unis), 2005. 42

[BSG98] Laure BRISOUX, Lakhdar SAÏS, et Éric GRÉGOIRE. « Mieux Exploiter les échecs au sein des arbres de recherche à la Davis et Putnam ». Dans *Actes des Quatrièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC'98)*, pages 31–39, Nantes (France), mai 1998. 16

[BSG01] Laure BRISOUX, Lakhdar SAÏS, et Éric GRÉGOIRE. « Validation of knowledge-based systems by means of stochastic search ». *International Journal of Intelligent Systems*, 16(3) :319–332, 2001. 47

[BW04] Fahiem BACCHUS et Jonathan WINTER. « Effective preprocessing with hyper-resolution and equality reduction ». Dans *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2929 de *Lecture Notes in Computer Science*, pages 341–355, Santa Margherita Ligure, Italy, mai 2003. Published in 2004. Springer. 34

[Cas97] Thierry CASTELL. « *Consistance et déduction en logique propositionnelle* ». Thèse de doctorat, Université Paul Sabatier, Toulouse, France, janvier 1997. 53

[CC96] Thierry CASTELL et Michel CAYROL. « Computation of prime implicates and prime implicants by the Davis and Putnam Procedure ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 61–64, Budapest (Hungary), août 1996. 48

[Che06] Wai-Kai CHEN. *The VLSI Handbook, Second Edition (Electrical Engineering Handbook)*. CRC Press, Inc., Boca Raton, FL, USA, 2006. 53

[Coo71] Stephen A. COOK. « The complexity of theorem-proving procedures ». Dans *STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM. 6, 31

[Cra93] James M. CRAWFORD. « Solving Satisfiability Problems Using a Combination of Systematic and Local Search ». Dans *Working notes of the Second Challenge on Satisfiability Testing organized by Center for Discrete Mathematics and Computer Science of Rutgers University*, octobre11-13 1993. http://dimacs.rutgers.edu/Challenges/. 21, 22

[CS88] Vašek CHVÁTAL et Endre SZEMERÉDI. « Many hard examples for resolution ». *Journal of the Association for Computing Machinery*, 35(4) :759–768, 1988. 16

[DABC96] Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD, et Jacques CARLIER. « SAT versus UNSAT ». Dans D.S. JOHNSON et M.A. TRICK, éditeurs, *Selected papers of Second DIMACS Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University*, volume 26 de *DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society*, pages 415–436, 1996. http://dimacs.rutgers.edu/Volumes/Vol26.html. 15, 16, 34, 35

[Dal96] Mukesh DALAL. « An Almost Quadratic Class of Satisfiability Problems ». Dans W. WAHLSTER, éditeur, *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'91)*, pages 355–359, Budapest (Hungary), août 1996. John Wiley & Sons, Ltd. 31

[DB96] Olivier DUBOIS et Yacine BOUFKHAD. « From very hard doubly balanced SAT formulae to easy unbalanced SAT formulae, variations of the satisfiability threshold ». Dans J.G. DING-ZHU DU et P. PARDALOS, éditeurs, *Proceedings of the First Worshop on Satisifiability (SAT'96)*, Siena, Italy, mars 1996. 16

[DBM00] Olivier DUBOIS, Yacine BOUFKHAD, et Jacques MANDLER. « Typical random 3-SAT formulae and the satisfiability threshold ». Dans *SODA '00 : Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 126–127, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. 16

[DD01] Olivier DUBOIS et Gilles DEQUEN. « A backbone-search heuristic for efficient solving of hard 3–SAT formulae ». Dans *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, volume 1, pages 248–253, Seattle, Washington, USA, août 4-10 2001. . . . . 15, 16

[DD04] Gilles DEQUEN et Olivier DUBOIS. « kcnfs : An Efficient Solver for Random k-SAT Formulae ». Dans *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2929 de *Lecture Notes in Computer Science*, pages 486–501, Santa Margherita Ligure, Italy, mai 2003. Published in 2004. Springer. 15, 56

[DDD+05a] Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, Bertrand MAZURE, Richard OSTROWSKI, et Lakhdar SAÏS. « Using Boolean Constraint Propagation for Sub-clause Deduction ». Dans *Proceedings of the Eleventh International Conference on Principles*

and Practice of Constraint Programming (CP'05), volume 3709 de *Lecture Notes in Computer Science*, pages 757–761, Sitges (Barcelona), Spain, octobre 2005. Springer. . . . 35

[DDD+05b] Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, Bertrand MAZURE, Richard OSTROWSKI, et Lakhdar SAÏS. « Utilisation de la Propagation de Contraintes pour la Production de Sous-Clauses ». Dans *Actes des Premières Journées Francophones de la Programmation par Contraintes (JFPC'05)*, pages 69–78, Lens, France, juin 2005. . 35

[DE92] Mukesh DALAL et David W. ETHERINGTON. « A hierarchy of tractable satisfiability problems ». *Information Processing Letters*, 44(4) :173–180, 10 décembre 1992. . . . 31

[dK89] Johan de KLEER. « A Comparison of ATMS and CSP Techniques. ». Dans *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 290–296, Detroit, Michigan, USA, 1989. 46

[dlBSW03] Maria Garcia de la BANDA, Peter J. STUCKEY, et Jeremy WAZNY. « Finding all Minimal Unsatisfiable Subsets ». Dans *Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDL'03)*, pages 32–43, 2003. . . . . 42, 44

[DLL62] Martin DAVIS, George LOGEMANN, et Donald LOVELAND. « A Machine Program for Theorem Proving ». *Journal of the Association for Computing Machinery*, 5 :394–397, 1962. . . . . 13, 14, 22

[DP60] Martin DAVIS et Hilary PUTNAM. « A Computing Procedure for Quantification Theory ». *Journal of the Association for Computing Machinery*, 7 :201–215, 1960. 14

[EB05] Niklas EÉN et Armin BIERE. « Effective Preprocessing in SAT Through Variable and Clause Elimination ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 61–75, St. Andrews, Scottland, juin 2005. Springer. 34

[EG92] Thomas EITER et Georg GOTTLOB. « On the complexity of propositional knowledge base revision, updates, and counterfactuals ». *Artificial Intelligence*, 57(2-3) :227–270, 1992. 9

[EG02] Thomas EITER et Georg GOTTLOB. « Hypergraph Transversal Computation and Related Problems in Logic and AI ». Dans *JELIA '02 : Proceedings of the European Conference on Logics in Artificial Intelligence*, volume 2424 de *lncs*, pages 549–564, London, UK, 2002. Springer-Verlag. 9

[ES04] Niklas EÉN et Niklas SÖRENSON. « An Extensible SAT-solver. ». Dans *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2929 de *Lecture Notes in Computer Science*, pages 333–336, Santa Margherita Ligure, Italy, mai 2003. Published in 2004. Springer. 15, 27

[FF04] Brian FERRIS et Jon FROEHLICH. « WalkSAT as an Informed Heuristic to DPLL in SAT Solving ». Rapport technique, CSE 573 : Artificial Intelligence, 2004. http://www.cs.washington.edu/homes/bdferris/papers/WalkSAT-DPLL.pdf.. 21, 22

[FGMS05] Olivier FOURDRINOY, Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Exploring Hybrid Algorithms for SAT ». Dans G. SUTCLIFFER et A. Voronkov (EDS), éditeurs, *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'05)*, pages 33–37, Montego Bay, Jamaïque, dec 2005. short paper. 21, 23

[FGMS07a] Olivier FOURDRINOY, Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Eliminating Redundant Clauses in Sat Instances ». Dans *Proceedings of The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, volume 4510 de *Lecture Notes in Computer Science*, pages 71–83, Brussels, Belgium, mai 23-26 2007. Springer. 31

[FGMS07b] Olivier FOURDRINOY, Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Reducing hard SAT instances to polynomial ones ». Dans *Proceedings of the 2007 IEEE international conference on Information Reuse and Integration (IEEE-IRI'07)*, pages 18–23, Las Vegas, NV, USA, août 2007. ..... 32, 87

[FH07] Lei FANG et Michael S. HSIAO. « A new hybrid solution to boost SAT solver performance ». Dans *DATE '07 : Proceedings of the conference on Design, automation and test in Europe*, pages 1307–1313, San Jose, CA, USA, 2007. EDA Consortium. 20, 21, 22

[FH08] Lei FANG et Michael S. HSIAO. « Boosting SAT Solver Performance via a New Hybrid Approach ». *Journal on Satisfiability, Boolean Modeling and Computation*, 5 :243–261, 2008. Research note. 20

[Fou07] Olivier FOURDRINOY. « *Utilisation de techniques polynomiales pour la résolution pratique d'instances de SAT* ». Thèse de doctorat, Université d'Artois, Faculté des Sciences Jean Perrin, Lens, France, novembre 2007. 23

[FR04] Hai FANG et Wheeler RUML. « Complete Local Search for Propositional Satisfiability ». Dans *Proceedings of the Nineteenth American National Conference on Artificial Intelligence (AAAI'04)*, pages 161–166, 2004. ..... 20, 21, 22, 27

[Fre95] Jon William FREEMAN. « *Improvements to Propositional Satisfiability Search Algorithms* ». Ph.d. thesis, University of Pennsylvania, Department of Computer and Information Science, 1995. ..... 15, 16

[Gin93] Matthew L. GINSBERG. « Dynamic Backtracking ». *Journal of Artificial Intelligence Research*, 1 :25–46, 1993. 44

[Glo89] Fred W. GLOVER. « Tabu search - Part I ». *ORSA Journal of Computing*, 1 :190–206, 1989. 20

[Glo90] Fred W. GLOVER. « Tabu search - Part II ». *ORSA Journal of Computing*, 2 :4–32, 1990. 20

[GM02] Éric GRÉGOIRE et Bertrand MAZURE. « About the incremental validation of first-order stratified knowledge-based decision-support systems ». *Information Sciences*, 142 :117–129, 2002. 47

[GMOS05] Éric GRÉGOIRE, Bertrand MAZURE, Richard OSTROWSKI, et Lakhdar SAÏS. « Automatic extraction of functional dependencies ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 de *Lecture Notes in Computer Science*, pages 122–132, Vancouver (BC) Canada, mai 10-13 2004. Revised selected papers published in 2005. Sringer. 29, 35, 53, 87

[GMP06a] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Extracting MUSes ». Dans *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'01)*, pages 387–391, Trento, Italy, aug 2006. 40

[GMP06b] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Tracking MUSes and Strict Inconsistent Covers ». Dans *Proceedings of the Sixth ACM/IEEE International Conference on Formal Methods in Computer Aided Design (FMCAD'06)*, pages 39–46, San Jose, California, USA, novembre 2006. 43

[GMP07a] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Boosting a Complete Technique to Find MSS and MUS thanks to a Local Search Oracle ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2300–2305, Hyderabad, India, janvier 6-16 2007. 42

[GMP07b] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Local-Search Extraction of MUSes ». *Constraints*, 12(3) :325–344, septembre 2007. . . . . 42, 87

[GMP07c] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « MUST : Provide a Finer-Grained Explanation of Unsatisfiability ». Dans Christian BESSIÈRE, éditeur, *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 de *Lecture Notes in Computer Science*, pages 317–331. Springer, septembre 2007. . . . . 45, 87

[GMP08a] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « On Approaches to Explaining Infeasibility of Sets of Boolean Clauses ». Dans *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, pages 74–83, Dayton, Ohio, USA, novembre 2008. 41

[GMP08b] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « On Finding Minimally Unsatisfiable Cores of CSPs ». *International Journal on Artificial Aintelligence Tools*, 17(4) :745–763, août 2008. 45

[GMP09a] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Does This Set of Clauses Overlap with at Least One MUS ? ». Dans *Proceedings of the 22nd International Conference on Automated Deduction (CADE-22*, pages 100–115, Montreal, P.Q., Canada, 2009. Springer-Verlag (Berlin, Heidelberg). 41

[GMP09b] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Using local search to find MSSes and MUSes ». *European Journal of Operational Research*, 199(3) :640–646, décembre 2009. . . . . 42, 87

[GMPS06] Éric GRÉGOIRE, Bertrand MAZURE, Cédric PIETTE, et Lakhdar SAÏS. « A New Heuristic-based albeit Complete Method to Extract MUCs from Unsatisfiable CSPs ». Dans *Proceedings of the IEEE International Conference on Information Reuse and Integration (IEEE-IRI'2006)*, pages 325–329, Waikoloa, Hawaii, USA, septembre 2006. 45

[GMR02] Jens GOTTLIEB, Elena MARCHIORI, et Claudio ROSSI. « Evolutionary algorithms for the satisfiability problem ». *Evol. Comput.*, 10(1) :35–50, 2002. 14

[GMS98] Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Logically-complete local search for propositional nonmonotonic knowledge bases ». Dans Ilkka NIEMELÄ et Torsten SCHAUB, éditeurs, *Proceedings of the Seventh International Workshop on NonMonotonic Reasoning (NMR'98)*, numéro 52 dans Series A : Research Reports, pages 37–45, Trento, Italy, 1998. Helsinky University of Technology Research Report, Digital Systems Laboratory. ISSN 0783 5396, ISBN 951 22 4072 6. 47

[GMS02] Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Using failed local search for SAT as an oracle for tackling harder A.I. problems more efficiently ». Dans D. SCOTT, éditeur, *Proceedings of the Tenth International Conference on Artificial Intelligence : Methodology, Systems, Applications (AIMSA'02)*, volume 2443 de *Lecture Notes in Computer Science*, pages 51–60, Varna, Bulgarie, septembre 2002. Springer Verlag. 47

[GMS09] Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Local autarkies searching for the dynamic partition of CNF formulae ». Dans *Proceedings of the 21st International*

Conference on Tools with Artificial Intelligence (ICTAI'09), page to appear, Newark, New Jersey, USA, novembre 2-5 2009. IEEE Computer Society. 33

[GN02] Eugene P. GOLDBERG et Yakov NOVIKOV. « BerkMin : a Fast and Robust SAT-Solver ». Dans *In Proceedings of Design Automation and Test in Europe (DATE'02)*, pages 142–149, Paris, 2002. 15

[GN07] Eugene GOLDBERG et Yakov NOVIKOV. « BerkMin : A fast and robust Sat-solver ». *Journal of Discrete Applied Mathematics*, 155(12) :1549–1561, 2007. 18

[GNT01] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « QuBE : A system for deciding Quantified Boolean Formulas Satisfiability ». Dans *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, volume 2083 de *Lecture Notes in Computer Science*, pages 364–369, Siena, Italy, June 2001. Springer. 48

[Gol79] Allen GOLDBERG. « On the Complexity of the Satisfiability Problem ». Rapport technique, New York University, 1979. . . . . 15, 16

[GS88] Giorgio GALLO et Maria Grazia SCUTELLÁ. « Polynomially solvable satisfiability problems ». *Information Processing Letters*, 29(5) :221–227, 24 novembre 1988. 31

[GSK98] Carla P. GOMES, Bart SELMAN, et Henry KAUTZ. « Boosting Combinatorial Search through Randomization ». Dans *Proceedings of the Fifteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, Madison, Wisconsin, USA, juillet 1998. American Association for Artificial Intelligence Press. 17

[GU89] Giorgio GALLO et Giampaolo URBANI. « Algorithms for Testing the Satisfiability of Propositional Formulae. ». *Journal of Logic Programming*, 7(1) :45–61, juillet 1989. 36

[Gu92] Jun GU. « Efficient local search for very large-scale satisfiability problems ». *SIGART Bulletin*, 3(1) :8–12, janvier 1992. 19

[Hao95] Jin-kao HAO. « A Clausal Genetic Representation and Its Evolutionary Procedures for Satisfiability Problems ». Dans *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 289–292. Springer-Verlag, avril 1995. 14

[HD04] William S. HAVENS et Bistra N. DILKINA. « A Hybrid Schema for Systematic Local Search ». Dans *Proceedings of The Seventeenth Canadian Conference on Artificial Intelligence (AI'2004)*, volume 3060 de *Lecture Notes in Computer Science*, pages 248–260. Springer, 2004. 21

[Héb08] Jean-Jacques HÉBRARD. « *Classes polynomiales* », Chapitre 2, pages 73–98. Dans « Problème SAT : progrès et défis » de Saïs [Saï08], mai 2008. 27

[HJS08] Youssef HAMADI, Saïd JABBOUR, et Lakhdar SAÏS. « ManySat : solver description ». Rapport technique MSR-TR-2008-83, Microsoft Research, 2008. SAT-race 2008. . . . 55

[HJS09a] Youssef HAMADI, Said JABBOUR, et Lakhdar SAIS. « Control-based Clause Sharing in Parallel SAT Solving ». Dans *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)*, pages 499–504, Pasadena, California, USA, juillet 11-17 2009. 55

[HJS09b] Youssef HAMADI, Said JABBOUR, et Lakhdar SAIS. « ManySAT : a Parallel SAT Solver ». *Journal on Satisfiability, Boolean Modeling and Computation*, 6 :245–262, 2009. . 55

[HK05] Edward A. HIRSCH et Arist KOJEVNIKOW. « UnitWalk : A new SAT solver that uses local serach guided by unit clause elimination ». *Annals of Mathematics and Artificial Intelligence*, 43(1-4) :91–111, 2005. 20

[HK08] Anthony HUNTER et Sébastien KONIECZNY. « Measuring Inconsistency through Minimal Inconsistent Sets ». Dans Gerhard BREWKA et Jérôme LANG, éditeurs, *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR'08)*, pages 358–366, Sydney, Australia, 2008. American Association for Artificial Intelligence Press. 41

[HL99] Benjamin HAN et Shie-Jue LEE. « Deriving minimal conflict sets by CS-Trees with mark set in diagnosis from first principles ». Dans *IEEE Transactions on Systems, Man, and Cybernetics*, volume 29, pages 281–286, 1999. 44

[HLDV02] Djamal HABET, Chu Min LI, Laure DEVENDEVILLE, et Michel VASQUEZ. « A Hybrid Approach for SAT ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 de *Lecture Notes in Computer Science*, pages 19–24. Springer, septembre 2002. . . . . 20, 21, 22

[HLSB06] Frédéric HÉMERY, Christophe LECOUTRE, Lakhdar SAÏS, et Frédéric BOUSSEMART. « Extracting MUCs from Constraint Networks ». Dans *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'01)*, pages 113–117, Trento (Italie), 2006. . . . . 44, 45

[HM09] Pierre HANSEN et Nenad MLADENOVIC. « *Variable Neighborhood Search Methods* », pages 3975–3989. Springer, 2009. Encyclopedia of Optimization, Second Edition. Christodoulos A. Floudas and Panos M. Pardalos (eds). 52

[Hoo88] John N. HOOKER. « A quantitative approach to logical inference ». *Decision Support Systems*, 4(1) :45–69, 1988. 13

[Hoo99] Holger H. HOOS. « On the run-time behaviour of stochastic local search algorithms for SAT ». Dans *Proceedings of the Sixteenth American National Conference on Artificial Intelligence (AAAI'99)*, pages 661–666, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence. 20

[Hoo02] Holger H. HOOS. « An adaptive noise mechanism for walkSAT ». Dans *Proceedings of the Eighteenth American National Conference on Artificial Intelligence (AAAI'02)*, pages 655–660, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. 20

[Hor51] Alfred HORN. « On sentences which are true of direct unions of algebras ». *Journal of Symbolic Logic*, 16 :14–51, 1951. 31

[HTH02] Frank HUTTER, Dave A. D. TOMPKINS, et Holger H. HOOS. « Scaling and Probabilistic Smoothing : Efficient Dynamic Local Search for SAT ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 de *Lecture Notes in Computer Science*. Springer, septembre 2002. . 20, 27

[Hua05] Jinbo HUANG. « MUP : A Minimal Unsatisfiability Prover ». Dans *Proceedings of the Tenth Asia and South Pacific Design Automation Conference (ASP-DAC-05)*, pages 432–437, Shanghai, China, 2005. 40

[Hua07] Jinbo HUANG. « The Effect of Restarts on the Efficiency of Clause Learning ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2318–2323, Hyderabad, India, janvier 6-16 2007. 17

[Iwa89] Kazuo IWAMA. « CNF satisfiability test by counting and polynomial average time ». *SIAM Journal on Computing*, 18(2) :385–391, 1989. 13

[Jab08] Saïd JABBOUR. « *De la satisfiabilité é propositionnelle aux formules booléennes quantifiées* ». Thèse de doctorat, Université d'Artois, Faculté des Sciences, Lens, France, 5 décembre 2008. 49

[JB00] Narendra JUSSIEN et Vincent BARICHARD. « The PaLM system : explanation-based constraint programming ». Dans *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP'00*, pages 118–133, Singapour, septembre 2000. 44

[JDB00] Narendra JUSSIEN, Romuald DEBRUYNE, et Patrice BOIZUMAULT. « Maintaining Arc-Consistency within Dynamic Backtracking ». Dans *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP'00)*, volume 1894 de *Lecture Notes in Computer Science*, pages 249–261, Singapore, septembre 2000. Springer. 44

[JL02] Narendra JUSSIEN et Olivier LHOMME. « Local Search with Constraint Propagation and Conflict-Based Heuristics ». *Artificial Intelligence*, 139(1) :21–45, juillet 2002. . 21, 22

[JOR88] Serge JEANNICOT, Laurent OXUSOFF, et Antoine RAUZY. « Évaluation sémantique en calcul propositionnel : une propriété de coupure pour rendre plus efficace la procédure de Davis et Putnam ». *Revue d'Intelligence Artificielle*, 2(1) :41–60, 1988. 33

[JSD96] Brigitte JAUMARD, Mihnea STAN, et Jacques DESROSIERS. « Tabu Search and a quadratic relaxation for the satisfiability problem ». Dans D.S. JOHNSON et M.A. TRICK, éditeurs, *Selected papers of Second DIMACS Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University*, volume 26 de *DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society*, pages 457–478, 1996. http://dimacs.rutgers.edu/Volumes/Vol26.html. 19

[Jun01] Ulrich JUNKER. « QuickXplain : Conflict Detection for Arbitrary Constraint Propagation Algorithms ». Dans Christian BESSIÈRE, éditeur, *Workshop on Modelling and Solving problems with constraints (CONS-1) suring Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001. http://www.lirmm.fr/~bessiere/proc_wsijcai01.html. 44

[Jun04] Ulrich JUNKER. « QuickXplain : Preferred Explanations and Relaxations for Over-Constrained Problems ». Dans *Proceedings of the Nineteenth American National Conference on Artificial Intelligence (AAAI'04)*, pages 167–172, 2004. 44

[JW90] Robert G. JEROSLOW et Jinchang WANG. « Solving Propositional Satisfiability Problems ». *Annals of Mathematics and Artificial Intelligence*, 1 :167–187, 1990. . . . . 13, 15, 16

[KK71] Robert A. KOWALSKI et Donald KUEHNER. « Linear Resolution with Selection Function ». *Artificial Intelligence*, 2 :227–260, 1971. 13

[KMS97] Henry A. KAUTZ, David MCALLESTER, et Bart SELMAN. « Exploiting variable Dependency in Local Search ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, Nagoya, Japan, août 23-29 1997. This abstract appears in "Abstracts of the Poster Sessions". 28

[Knu90] Donald E. KNUTH. « Nested Satisfiability ». *Acta Informatica*, 28 :1–6, 1990. 31

[KS02] Victor N. KRAVETS et Karem A. SAKALLAH. « Resynthesis of multi-level circuits under tight constraints using symbolic optimization ». Dans *Proceedings of the 2002 IEEE/ACM*

international conference on Computer-aided design, pages 687–693, San Jose, California, 2002. ACM, New York, NY, USA. 53

[KS03] Henri KAUTZ et Bart SELMAN. « Ten Challenges Redux : Recent Progress in Propositional Reasoning and Search ». Dans *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833 de *Lecture Notes in Computer Science*, pages 1–18, Kinsale, Ireland, septembre 2003. Springer. 14

[Kul96] Oliver KULLMANN. Worst-case Analysis, 3-SAT Decision and Lower Bounds : Approaches for Improved SAT Algorithms. Dans Dingzhu DU, Jun GU, et Panos M. PARDALOS, éditeurs, *Workshop on Satisfiability Problem : Theory and Applications*, volume 35 de *DIMACS Series in Discrete Mathematics and Theorical Computer Science*, pages 261–314. American Mathematical Society, DIMACS Center, Rutgers (State University of New Jersey, USA), décembre 1996. 34

[Kul00] Oliver KULLMANN. « Investigations on Autark Assignments ». *Journal of Discrete Applied Mathematics*, 107(1-3) :99–137, 2000. . . . . 27, 33

[LA97] Chu Min LI et ANBULAGAN. « Heuristics Based on Unit Propagation for Satisfiability Problems ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya, Japan, août 23-29 1997. . . 15, 16, 28, 34, 35

[Lab00] François LABURTHE. « CHOCO : implementing a CP kernel ». Dans *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP'00*, Singapour, 2000. http://www.choco-constraints.net. 44

[Lew78] Harry R. LEWIS. « Renaming a Set of Clauses as Horn Set ». *Journal of the Association for Computing Machinery*, 25 :134–135, 1978. 31

[LH05] Chu Min LI et Wen Qi HUANG. « Diversification and Determinism in Local Search for Satisfiability ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 158–172, St. Andrews, Scottland, juin 2005. Springer. 20

[Li00] Chu Min LI. « Integrating equivalency reasoning into Davis-Putnam procedure ». Dans *Proceedings of the Seventeenth American National Conference on Artificial Intelligence (AAAI'00)*, pages 291–296, 2000. . . . . 15, 28

[Lib05] Paolo LIBERATORE. « Redundancy in logic I : CNF propositional formulae ». *Artificial Intelligence*, 163(2) :203–232, 2005. . . . . 27, 30

[Lib06] Paolo LIBERATORE. « Complexity results on DPLL and resolution ». *ACM Transactions on Computational Logic*, 7(1) :84–107, 2006. 15

[Lib08a] Paolo LIBERATORE. « Redundancy in logic II : 2CNF and Horn propositional formulae ». *Artificial Intelligence*, 172(2-3) :265–299, 2008. . . . . 27, 30

[Lib08b] Paolo LIBERATORE. « Redundancy in logic III : Non-monotonic reasoning ». *Artificial Intelligence*, 172(11) :1317–1359, 2008. . . . . 27, 30

[LMS04] Inês LYNCE et João P. MARQUES-SILVA. « On Computing Minimum Unsatisfiable Cores ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver (BC) Canada, 2004. 41

[LMS08] Florian LETOMBE et João P. MARQUES-SILVA. « Improvements to Hybrid Incremental SAT Algorithms ». Dans *Proceedings of the Eleventh International Conference on*

Theory and Applications of Satisfiability Testing (SAT'08), volume 4996 de *Lecture Notes in Computer Science*, pages 161–181, Guangzhou, P. R. China, mai 2008. Springer.. 20, 21, 22

[Lov78] Donald W. LOVELAND. *Automated theorem proving : A logical basis (Fundamental studies in computer science)*. Elsevier, 1978. 13

[LS05] Mark H. LIFFITON et Karem A. SAKALLAH. « On Finding All Minimally Unsatisfiable Subformulas ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 173–186, St. Andrews, Scottland, juin 2005. Springer. 42

[LS08] Mark H. LIFFITON et Karem A. SAKALLAH. « Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints ». *Journal of Automated Reasoning*, 40(1) :1–33, janvier 2008. 42

[LSB07] Matthew LEWIS, Tobias SCHUBERT, et Bernd BECKER. « Multithreaded SAT Solving ». Dans *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 926–931, Washington, DC, USA, 2007. IEEE Computer Society. 55

[LSH06] Frédéric LARDEUX, Frédéric SAUBION, et Jin-Kao HAO. « GASAT : A Genetic Local Search Algorithm for the Satisfiability Problem ». *Evolutionary Computation*, 14(2) :223–253, 2006. 14

[LSZ93] Michael LUBY, Alistair SINCLAIR, et David ZUCKERMAN. « Optimal speedup of Las Vegas algorithms ». *Information Processing Letters*, 47(4) :173–180, 1993. 17

[LWZ07] Chu Min LI, Wanxia WEI, et Harry ZHANG. « Combining Adaptive Noise and Look-Ahead in Local Search for SAT ». Dans *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 de *Lecture Notes in Computer Science*, pages 121–133, Lisbon, Portugal, 2007. Springer. . . . 20, 27

[MH97] Nenad MLADENOVIC et Pierre HANSEN. « Variable neighborhood search ». *Computers & OR*, 24(11) :1097–1100, 1997. 52

[MLA+05] Maher MNEIMNEH, Inês LYNCE, Zaher ANDRAUS, João MARQUES-SILVA, et Karem SAKALLAH. « A Branch-and-Bound Algorithm for Extracting Smallest Minimal Unsatisfiable Formulas ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 467–474, St. Andrews, Scottland, juin 2005. Springer. 41

[MM96] Bertrand MAZURE et Pierre MARQUIS. « Theory Reasoning within Implicant Cover Compilations ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 65–69, Budapest (Hungary), août 1996. 48

[MMZ+01] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG, et Sharad MALIK. « Chaff : engineering an efficient SAT solver ». Dans *DAC '01 : Proceedings of the 38th annual Design Automation Conference*, pages 530–535, New York, NY, USA, juin 2001. ACM. 13, 15, 16

[Mor93] Paul MORRIS. « The Break Out Method For Escaping From Local Minima ». Dans *Proceedings of the Eleventh American National Conference on Artificial Intelligence (AAAI'93)*, pages 40–45, 1993. . . . . 19, 20, 23

[MPZ02] Marc MÉZARD, Giorgio PARISI, et Riccardo ZECCHINA. « Analytic and Algorithmic Solution of Random Satisfiability Problems ». *Science*, 297 :812–815, 2002. 20

[MS85] Burkhard MONIEN et Ewald SPECKENMEYER. « Solving Satisfiability in Less Than 2^n Steps ». *Journal of Discrete Applied Mathematics*, 10 :287–295, 1985. . . . . 27, 33

[MSG95] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « TWSAT : a new local search algorithm for SAT. Performance and Analysis ». Dans *Proceedings of Constraint Programming Workshop on Solving Really Hard Problems*, pages 127–130, Cassis (France), septembre 1995. 20

[MSG97a] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Checking Several Forms of Consistency in Non-Monotonic Knowledge-Bases ». Dans Dov GABBAY, Rudolf KRUSE, Andreas NONNENGART, et Hans Jüergen OHLBACH, éditeurs, *Proceedings of the First International Joint Conference on Qualitative and Quantitative Pratical Reasoning (ECSQARU-FAPR'97)*, volume 1244 de *Lecture Notes in Artificial Intelligence*, pages 122–130, Bad Honnef (Germany), juin 1997. Springer. ISBN : 3-540-63095-3. 47

[MSG97b] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « An Efficient Technique to ensure the Logical Consistency of Interacting Knowledge Bases ». *International Journal of Cooperative Information Systems*, 6(1) :27–36, 1997. 47

[MSG97c] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Tabu Search for SAT ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, Providence (Rhode Island, USA), juillet 1997. . . . 20, 52, 87

[MSG98] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Boosting Complete Techniques thanks to local search methods ». *Annals of Mathematics and Artificial Intelligence*, 22 :319–331, 1998. . . . . 21, 22, 40, 87

[MSK97] David A. MCALLESTER, Bart SELMAN, et Henry A. KAUTZ. « Evidence for Invariants in Local Search ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, août 1997. 20

[MSS96] João P. MARQUES-SILVA et Karem A. SAKALLAH. « GRASP—a new search algorithm for satisfiability ». Dans *ICCAD '96 : Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society. 13, 15, 18

[MT02] Jakob MAUSS et Mugur TATAR. « Computing Minimal Conflicts for Rich Constraint Languages ». Dans *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI'97)*, pages 151–155, Lyon, France, 2002. 44

[MZ02] Marc MÉZARD et Riccardo ZECCHINA. « The random K-satisfiability problem : from an analytic solution to an efficient algorithm ». *Physical Review*, 66 :56–126, Novembre 2002. 20

[OGMS02] Richard OSTROWSKI, Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Recovering and exploiting structural knowledge from CNF formulas ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 de *Lecture Notes in Computer Science*, pages 185–199. Springer, septembre 2002. . . . . 29, 34, 53, 87

[OMA+04] Yoonna OH, Maher N. MNEIMNEH, Zaher S. ANDRAUS, Karem A. SAKALLAH, et Igor L. MARKOV. « AMUSE : a minimally-unsatisfiable subformula extractor ». Dans *Proceedings of the 41st annual conference on Design automation (DAC'04)*, pages 518–523, New York (États-Unis), 2004. ACM Press. 40

[OMS03] Richard OSTROWSKI, Bertrand MAZURE, et Lakhdar SAÏS. « LSAT Solver v2.0 ». Rapport technique, CRIL, 2003. http://www.cril.univ-artois.fr/spip/spip.php?article22. . . . 56

[OMSG03] Richard OSTROWSKI, Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Eliminating redundancies in SAT search trees ». Dans *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, pages 100–104, Scramento, California, USA, 2003. IEEE Computer Society (Washington, DC, USA). 36, 37

[Ost04] Richard OSTROWSKI. « *Reconnaissance et exploitation de propriétés structurelles pour la résolution du probléme SAT* ». Thèse doctorat, Université d'Artois, Faculté des Sciences Jean Perrin, Lens, France, décembre 2004. 30

[OU09] Kei OHMURA et Kazunori UEDA. « c-sat : A Parallel SAT Solver for Clusters ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 de *Lecture Notes in Computer Science*, pages 524–537, Swansea, Wales, United Kingdom, 2009. Springer. 55

[PHL08] Cédric PIETTE, Youssef HAMADI, et Saïs LAKHDAR. « Vivifying propositional clausal formulae ». Dans *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'03)*, pages 525–529, Patras (Greece), juillet 2008. 36

[Pie07] Cédric PIETTE. « *Techniques algorithmiques pour l'extraction de formules minimales inconsistantes* ». Thèse de doctorat, Université d'Artois, Faculté des Sciences Jean Perrin, Lens, France, novembre 2007. 40

[Pie08] Cédric PIETTE. « Let the Solver Deal with Redundancy ». Dans *Proceedings of 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, pages 67–73, Dayton, Ohio, USA, 2008. IEEE Computer Society (Washington, DC, USA). 31

[Pro93] Patrick PROSSER. « Hybrid algorithms for the constraint satisfaction problems ». *Computational Intelligence*, 9(3) :268–299, 1993. 44

[PTS07] Duc Nghia PHAM, John THORNTON, et Abdul SATTAR. « Building Structure into Local Search for SAT ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2359–2364, Hyderabad, India, janvier 6-16 2007. 30, 53

[Puc90] Douglas PUCKNELL. *Fundamentals of digital logic design : with VLSI circuit applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. 53

[Pur84] Paul W. PURDOM. « Solving satisfiability with less searching ». *IEEE transactions on pattern analysis ans machine intelligence*, PAMI-6(4) :510–513, juillet 1984. 36

[PW88] Christos H. PAPADIMITRIOU et David WOLFE. « The complexity of facets resolved ». *J. Comput. Syst. Sci.*, 37(1) :2–13, 1988. 9

[Rob65] John Alan ROBINSON. « A machine-oriented logic based on the resolution principle ». *Journal of the Association for Computing Machinery*, 12 :23–41, 1965. 13

[RSB99] Antoine RAUZY, Lakhdar SAÏS, et Laure BRISOUX. « Calcul propositionnel : vers une extension du formalisme ». Dans *Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99)*, pages 189–198, Lyon, 1999.. 28

[Saï08] Lakhdar SAÏS, éditeur. *Problème SAT : progrès et défis*. Collection Programmation Par Contraintes. Hermes, mai 2008. ..... 6, 234

[SBK05] Tian SANG, Paul BEAME, et Henry A. KAUTZ. « Heuristics for Fast Exact Model Counting ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, St. Andrews, Scottland, juin 2005. Springer. 18

[Sch96] Robert C. SCHRAG. « Compilation for critically constrained knowledge bases ». Dans *Proceedings of the Thirteenth American National Conference on Artificial Intelligence (AAAI'96)*, pages 510–515, juillet 1996. 48

[Sha40] Claude Elwood SHANNON. « A symbolic analysis of relay and switching circuits ». Thesis (m.s.), Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1940. 13

[Sha53] Lloyd S. SHAPLEY. A value for n-person games. Dans H. W. KUHN et A. W. TUCKER, éditeurs, *Contributions to the Theory of Games II*, numéro 28 dans Annals of Mathematics Studies (AM-28), pages 303–317. Princeton University Press, Princeton, New Jersey, USA, 1953. 41

[SKC93] Bart SELMAN, Henry A. KAUTZ, et Bram COHEN. « Local Search Strategies for Satisfiability Testing ». Dans *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993. 20

[SKC94] Bart SELMAN, Henry A. KAUTZ, et Bram COHEN. « Noise strategies for improving local search ». Dans *Proceedings of the Twelfth American National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, 1994. . . . . 20, 22, 27

[SKM97] Bart SELMAN, Henry A. KAUTZ, et David A. MCALLESTER. « Ten Challenges in Propositional Reasoning and Search ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 50–54, Nagoya, Japan, août 23-29 1997. . . . . 14, 22, 25, 28

[SLB05] Tobias SCHUBERT, Matthew LEWIS, et Bernd BECKER. « PaMira - A Parallel SAT Solver with Knowledge Sharing ». Dans *Proceedings of the Sixth International Workshop on Microprocessor Test and Verification*, pages 29–36, Washington, DC, USA, 2005. IEEE Computer Society. 55

[SLM92] Bart SELMAN, Hector J. LEVESQUE, et David MITCHELL. « GSAT : A New Method for Solving Hard Satisfiability Problems ». Dans *Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992. . . . . 13, 19, 20

[SM08] Daniel SINGER et Anthony MONNET. « JaCk-SAT : A New Parallel Scheme to Solve the Satisfiability Problem (SAT) Based on Join-and-Check ». Dans Roman WYRZYKOWSKI, Jack DONGARRA, Konrad KARCZEWSKI, et Jerzy WASNIEWSKI, éditeurs, *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM'07) - Revised Selected Papers*, volume 4967 de *Lecture Notes in Computer Science*, pages 249–258, Gdansk, Poland, septembre 9-12 2007 2008. Springer. 55

[SP05] Sathiamoorthy SUBBARAYAN et Dhiraj K. PRADHAN. « NiVER : Non Increasing Variable Elimination Resolution for Preprocessing SAT Instances ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 de *Lecture Notes in Computer Science*, pages 276–291, Vancouver (BC) Canada, mai 10-13 2004. Revised selected papers published in 2005. Sringer. 34

[SV94] Thomas SCHIEX et Gérard VERFAILLIE. « Stubbornness : an enhancement scheme for Backjumping and Nogood Recording ». Dans *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'89)*, pages 165–169, Amsterdam, Netherlands, 1994. 45

[SV06] Daniel SINGER et Alain VAGNER. « Parallel Resolution of the Satisfiability Problem (SAT) with OpenMP and MPI ». Dans Roman WYRZYKOWSKI, Jack DONGARRA, Norbert MEYER, et Jerzy WASNIEWSKI, éditeurs, *Proceedings of the 6th International*

Conference on Parallel Processing and Applied Mathematics (PPAM'05) - Revised Selected Papers, volume 3911 de *Lecture Notes in Computer Science*, pages 380–388, Poznan, Poland, septembre 11-14 2005 2006. Springer. 55

[Tov84] Craig A. TOVEY. « A Simplified NP-complete Satisfiability Problem ». *Journal of Discrete Applied Mathematics*, 8 :85–89, 1984. 31

[US94] Tomás E. URIBE et Mark E. STICKEL. « Ordered Binary Decision Diagrams and the Davis-Putnam Procedure ». Dans *CCL '94 : Proceedings of the First International Conference on Constraints in Computational Logics*, pages 34–49, London, UK, 1994. Springer-Verlag. 14

[VG99] Allen VAN GELDER. « Autarky Pruning in Propositional Model Elimination Reduces Failure Redundancy ». *Journal of Automated Reasoning*, 23(2) :137–193, 1999. . 27, 33

[vMW08] Hans van MAAREN et Siert WIERINGA. « Finding Guaranteed MUSes Fast ». Dans *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 de *Lecture Notes in Computer Science*, pages 291–304, Guangzhou, P. R. China, mai 2008. Springer. 40

[WGS03] Ryan WILLIAMS, Carla P. GOMES, et Bart SELMAN. « Backdoors To Typical Case Complexity ». Dans *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178, Acapulco, Mexico, août 9-15 2003. . . 27, 29

[WvM99] Joost P. WARNERS et Hans van MAAREN. « A two phase algorithm for solving a class of hard satisfiability problems ». *Operations Research Letters*, 23(3–5) :81–88, 1999. . . 28

[XHHLB08] Lin XU, Frank HUTTER, Holger H. HOOS, et Kevin LEYTON-BROWN. « SATzilla : portfolio-based algorithm selection for SAT ». *Journal of Artificial Intelligence Research*, 32(1) :565–606, 2008. 56

[YD83] Susumu YAMASAKI et Shuji DOSHITA. « The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic ». *Journal of Information and Computation*, 59 :1–12, 1983. 31

[Zha05] Lintao ZHANG. « On subsumption removal and on-the-fly CNF simplification ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 482–489, St. Andrews, Scottland, juin 2005. Springer. 42

[ZLS06] Jianmin ZHANG, Sikun LI, et Shengyu SHEN. « Extracting Minimum Unsatisfiable Cores with a Greedy Genetic Algorithm ». Dans *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence (AI'06)*, volume 4304 de *Lecture Notes in Computer Science*, Hobart, Australia, décembre 2006. Springer. 41

[ZM02] Lintao ZHANG et Sharad MALIK. « The Quest for Efficient Boolean Satisfiability Solvers ». Dans *CADE-18 : Proceedings of the 18th International Conference on Automated Deduction*, pages 295–313, London, UK, 2002. Springer-Verlag. 18

[ZM03] Lintao ZHANG et Sharad MALIK. « Extracting small unsatisfiable cores from unsatisfiable boolean formula ». Dans *Sixth international conference on theory and applications of satisfiability testin g (SAT'03)*, Portofino (Italie), 2003. 40

[ZMMM01] Lintao ZHANG, Conor F. MADIGAN, Matthew H. MOSKEWICZ, et Sharad MALIK. « Efficient conflict driven learning in a boolean satisfiability solver ». Dans *ICCAD '01 : Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285, Piscataway, NJ, USA, novembre 2001. IEEE Press. 18

[ZZ96] Jian ZHANG et Hantao ZHANG. « Combining Local Search and Backtracking Techniques for Constraint Satisfaction ». Dans *Proceedings of the Thirteenth American National Conference on Artificial Intelligence (AAAI'96)*, pages 369–374, juillet 1996. 20, 21, 22

Résumé

Ce mémoire présente une synthèse des travaux de recherche que j'ai accomplis depuis ma thèse, il y a maintenant près de 10 ans. Mes activités de recherche ont pour objet la résolution pratique du problème NP-complet de référence, à savoir SAT, et de problèmes liés à celui-ci.

Le premier chapitre présente le contexte de mes travaux, décrit les problèmes abordés et définit les termes et les notations les plus utilisés dans ce mémoire. Les deux chapitres suivants traitent principalement de mes contributions : le premier développe mes travaux sur SAT et le second mes travaux sur des problèmes connexes à SAT. Sur SAT, après un bref état de l'art, je décris mes travaux autour de trois thèmes : l'hybridation de solveurs, l'exploitation de propriétés structurelles et les techniques de simplification. Le chapitre intitulé « Au-delà de SAT » regroupe mes contributions sur des problèmes tels que la recherche de noyaux incohérents dans le cadre propositionnel et dans le cadre plus général de réseaux de contraintes, la résolution de problèmes exprimés à l'aide de logiques non monotones et du premier ordre, la compilation de bases de connaissances, la résolution des formules booléennes quantifiées. Dans le dernier chapitre, j'énonce quelques perspectives de recherche.

L'ensemble de cette synthèse est comme de coutume suivi d'un curriculum vitae et d'une sélection de publications.

Mots-clés: SAT, Hybridation, Noyaux incohérents, CSP, QBF.