UNIVERSITÉ D'ARTOIS

Contributions à la résolution du problème de la Satisfiabilité Propositionnelle

THÈSE

présentée et soutenue publiquement le 03 octobre 2014

en vue de l'obtention du

Doctorat de l'Université d'Artois

Spécialité : Informatique

par

Jerry Garvin Lonlac Konlac

Composition du jury

Rapporteurs :	Chu Min Li	Université de Picardie Jules Verne
	Frédéric Saubion	Université d'Angers
Examinateur :	Arab Ali Chérif	Université de Paris 8
Co-Encadreurs :	Clémentin Tayou	Université de Dschang-Cameroun
	Said Jabbour	Université d'Artois
Directeur de Thèse :	Lakhdar Saïs	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

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Remerciements

Je dédie cette thèse
à ma famille.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Introduction générale

Le problème de satisfiabilité (SAT) en logique propositionnelle capture l'essence de nombreux problèmes difficiles présents dans de nombreuses disciplines telles que l'intelligence artificielle, la recherche opérationnelle, etc. C'est un problème de décision qui consiste à déterminer si une formule propositionnelle mise sous forme normale conjonctive admet ou non un modèle. La simplicité de l'énoncé et du langage propositionnel sous-jacent cache une difficulté théorique majeure - SAT est le premier problème à avoir été montré complet pour la classe NP (Non déterministe Polynomial) Cook (1971) - lui donnant ainsi une place centrale en théorie de la complexité et rendant son intérêt pratique encore plus important. On le retrouve au centre de nombreuses applications d'intérêts majeurs comme la vérification de matériels et de logiciels, la configuration de produit, la déduction automatique, la bioinformatique, la planification, et la cryptographie. En effet, par sa simplicité, la logique propositionnelle permet de représenter une grande variété de problèmes complexes et surtout d'étudier plus finement les algorithmes de résolutions associés. C'est donc, un langage de représentation des connaissances qui offre l'un des meilleurs compromis entre la puissance d'expression et l'efficacité algorithmique, vieux problème en représentation des connaissances. Cette affirmation énoncée souvent par nos prédécesseurs du domaine est aujourd'hui validée par les progrès algorithmiques spectaculaires réalisés dans le cadre de la résolution pratique du problème de la satisfiabilité propositionnelle. La modélisation d'applications du monde réel en logique propositionnelle et l'extension des résultats de SAT à des problématiques autour de SAT est devenue aujourd'hui une pratique courante. Cette nouvelle situation suscite un réel débat sur le fait que la complexité au pire cas généralement privilégié par la théorie de la complexité est loin de refléter la réalité de la résolution de problèmes en pratique Vardi (2010). C'est dans ce contexte et cette dynamique que se situe cette thèse.

Ce contexte de recherche active ouvre de nombreuses perspectives et pose de nouveaux défis à relever pour consolider et asseoir dans le temps ces avancés, sachant que la taille et la complexité des problèmes ou des nouvelles applications émergentes évoluent sensiblement. Ces défis émergent naturellement en analysant finement ces progrès au niveau de l'efficacité algorithmique. En effet, les compétitions internationales des prouveurs SAT organisées chaque année montrent d'une part (1) qu'une classe de problèmes avec des millions de clauses (ou contraintes) et des millions de variables peuvent être résolus efficacement et (2) elles mettent également en évidence toute une classe d'instances qu'aucun algorithme connu à l'heure actuel ne peut résoudre.

De manière générale, l'objectif de cette thèse est de proposer de nouvelles améliorations algorithmiques capablent de résoudre ces problèmes difficiles et ouverts

afin d'élargir la classe des problèmes SAT que l'on peut résoudre en pratique et d'étudier les instances issues d'applications.

Ce manuscrit est organisé autour de deux grandes parties.

La première partie composée de 3 chapitres est consacrée à la présentation de l'état de l'art du problème SAT. Nous introduisons au premier chapitre quelques notions essentielles à la compréhension de ce manuscrit ; à savoir : les bases de la théorie de la complexité, la définition formelle du problème SAT, ainsi que la notion de classes d'instances. Dans le deuxième chapitre, nous décrivons les principales approches de résolution du problème SAT. Finalement, avant de conclure cette partie, nous détaillons les différents composants des solveurs SAT modernes : les solveurs CDCL dans le troisième chapitre. Nous présentons également à la fin de ce chapitre quelques approches utilisées pour la résolution de SAT dans un cadre parallèle.

La deuxième partie contenant 4 chapitres et est essentiellement consacré à la présentation des différentes contributions que nous avons apportées dans le cadre de la résolution du problème SAT.

Notre première contribution (Chapitre 4) propose une technique originale de substitution dynamique de fonctions booléennes. Cette approche consiste à effectuer dans une phase de prétraitement, la reconnaissance d'un ensemble de fonctions booléennes à partir d'une formule booléenne sous forme normale conjonctive (CNF). Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises en substituant les arguments d'entrée par les arguments de sortie. Ceci conduit à une façon originale d'intégrer une forme restreinte de résolution étendue Tseitin (1968), l'un des plus puissants systèmes de preuve par résolution. Des expérimentations montrent la faisabilité de notre approche sur certaines classes d'instances SAT prises des récentes compétitions SAT et SAT-Race.

La seconde contribution (Chapitre 5) présente une approche de résolution basée sur le principe d'intensification de la recherche dans les solveurs SAT modernes. Plus précisément, nous proposons une technique permettant de combiner deux des composantes les plus importantes des solveurs SAT modernes, à savoir le redémarrage et l'heuristique d'ordonnancement des variables. Cette combinaison permet au solveur d'intensifier la recherche et en même temps d'éviter le trashing, c'est à dire de visiter le même sous espace plusieurs fois. Des expérimentations que nous avons effectuées montrent que ce simple principe d'intensification apporte des améliorations significatives lorsqu'il est intégré aux solveurs SAT modernes. Ce travail nous a permis de montrer qu'il reste encore des possibilités d'améliorations des heuristiques d'ordonnancement des variables et que des études plus poussées sur le lien entre les redémarrages, les heuristiques d'ordonnancement des variables et l'apprentissage sont nécessaires.

Dans la troisième contribution (Chapitre 6), nous proposons un nouveau schéma d'apprentissage pour SAT. Cette approche consiste à traverser le graphe d'implications séparément à chaque conflit à partir des deux littéraux conflits x et $\neg x$ et à dériver une nouvelle classe de clauses Bi-Assertives Pipatsrisawat et Darwiche (2008a) qui peuvent conduire à un graphe d'implications plus compact. L'originalité de ces nouvelles clauses Bi-Assertives réside dans le fait qu'elles sont plus courtes et tendent à induire plus d'implications que les clauses Bi-Assertives classiques Pipatsrisawat et Darwiche (2008a). Les clauses Bi-Assertives classiques qui sont généralement dérivées en utilisant l'apprentissage traditionnel et les nouvelles clauses Bi-Assertives que nous proposons ici diffèrent sur certains aspects. Premièrement, les nouvelles clauses Bi-Assertives (respectivement clauses Bi-Assertives classiques) sont satisfaites (respectivement falsifiées) sous l'interprétation courante. Deuxièmement, les deux classes de clauses sont générées en utilisant un parcours différent du graphe d'implications.

Dans la quatrième contribution (Chapitre 7), nous révisitons les différentes stratégies de réduction de la base des clauses apprises utilisées dans les solveurs CDCL et proposons plusieurs autres nouvelles stratégies basées sur la pertinence des clauses apprises. Nous dérivons premièrement une simple stratégie de réduction, appelée "Size-Bounded Randomized Strategy" (abrégée SBR), qui conserve les courtes clauses (de taille inférieure ou égale à k), tout en supprimant aléatoirement les clauses de taille supérieure à k. La stratégie résultante est plus performante que celle de l'état de l'art, à savoir celle basée sur le LBD, sur les instances satisfiables prises de la dernière compétition SAT 2013. Convaincu par l'importance des courtes clauses, nous avons proposé plusieurs variantes dynamiques efficaces qui nous permettent de garder les clauses apprises qui sont plus susceptibles de couper des branches en haut de l'arbre de recherche. Ces travaux nous ont permis de montrer que les stratégies d'apprentissage basées sur la limitation d'une borne sur la taille, proposées il y a plus d'une quinzaine d'années Marques-Silva et Sakallah (1996a), Bayardo et Miranker (1996), Bayardo Jr. et Schrag (1997) restent de bonnes mesures pour prédire la qualité des clauses apprises. Ce travail nous a permis également de montrer que l'ajout de la randomisation à l'apprentissage par la limitation d'une borne sur la taille des clauses est un bon moyen de parvenir à une diversification contrôlée.

Finalement, une conclusion générale et de nombreuses perspectives ouvrant des voies de recherches futures viennent conclure ce manuscrit.

Première partie

Satisfiabilité propositionnelle : Modèles et Algorithmes

Introduction

Le problème SAT, problème de décision qui vise à savoir si une formule de la logique propositionnelle mise sous forme normale conjonctive possède une évaluation vrai, est le premier problème à avoir été montré NP-complet (Cook (1971)), il occupe un rôle central en théorie de la complexité. Le problème SAT est un problème générique, de nombreux problèmes provenant d'autres domaines tels que le problème de modèle checking, la déduction automatique, la planification, la bioinformatique *etc.* peuvent être réduits au problème SAT. Depuis plusieurs années, de nombreux algorithmes ont été proposés pour résoudre ce problème, la résolution (Robinson (1965), Galil (1977)), l'énumération (Quine (1950)), la procédure DPLL (Davis *et al.* (1962)), et avec l'avènement des solveurs SAT modernes : CDCL (*Conflict Driven, Clause Learning*), des instances industrielles de centaines de milliers de variables et de millions de clauses peuvent être résolues en quelques minutes.

Cette partie comprend trois chapitres. Le premier chapitre est consacré à la présentation du problème SAT. Nous commençons par introduire quelques notions de complexité afin de situer au mieux le problème SAT et son intérêt théorique. Ensuite, après avoir défini le formalisme de la logique propositionnelle, nous définissons le problème SAT et présentons quelques problèmes autours de SAT. Avant de conclure ce chapitre, nous présentons les différentes classes d'instances permettant d'évaluer les performances des prouveurs SAT.

Dans le deuxième chapitre, les méthodes de résolution sont introduites, nous présentons les différentes méthodes de résolution du problème SAT ainsi que les composantes qui ont conduit à l'élaboration des prouveurs SAT modernes.

Le dernier chapitre de cette partie est consacré au prouveur CDCL, nous y détaillons les principales composantes d'un solveur SAT moderne de type CDCL. Plus précisément, le redémarrage, l'analyse de conflits, la propagation unitaire et l'heuristique VSIDS sont présentés dans ce chapitre. Nous terminons par une brève description des principales approches utilisées pour la résolution du problème SAT en parallèle

Chapitre 1

Problématique, Définitions et Notations

Sommaire

CE CHAPITRE a pour objectif de fournir aux lecteurs les notions élémentaires de la logique propositionnelle. Nous commençons par l'introduction de quelques notions de complexité afin de situer au mieux l'importance théorique et pratique du problème de la satisfiabilité d'une formule propositionnelle (en d'autre mot le problème SAT). Ensuite, nous définissons de manière formelle ce qu'est le problème SAT. Après avoir definit quelques problèmes autours de SAT, Nous terminons ce chapitre par une revue des différents types de problèmes encodés à l'aide de la logique propositionnelle. Ces problèmes ainsi codés sont le plus souvent nommés instances et sont utilisés pour l'évaluation des prouveurs SAT.

1.1 Théorie de la complexité des algorithmes

Cette section a simplement pour objectif de situer le problème SAT dans la hiérarchie polynomiale. Après avoir situé la théorie de la complexité dans le contexte

historique, Nous donnons seulement des notions et propriétés nécessaires. Pour plus de détails sur la théorie de la complexité, des ouvrages Turing et Girard (1991), Papadimitriou (1994), Creignou *et al.* (2001) peuvent être consultés.

La théorie de la complexité a été introduite afin de permettre de mesurer la difficulté d'un problème algorithmique en fonction de la taille de ses instances. De manière plus générale, la théorie de la complexité mesure la difficulté d'un algorithme résolvant le problème considéré.

La complexité d'un algorithme pour résoudre un problème est une mesure des ressources nécessaires pour son exécution. Les ressources qui sont prises en compte sont le temps et l'espace mémoire informatique. La théorie de la complexité des algorithmes étudie formellement la quantité de ressources nécessaires pour l'exécution d'un algorithme ainsi que la difficulté intrinsèque des problèmes calculables. Plus précisément, l'objectif de cette théorie est d'étudier l'évolution des ressources nécessaires pour résoudre un problème en fonction de la taille des données fournies en entrée. Formellement, étant donnée un algorithme $\mathcal{A}$, la théorie de la complexité s'intéresse à estimer le temps ou l'espace mémoire nécessaire au calcul de $\mathcal{A}_{(n)}$ en fonction de n, où n est la taille de la donnée d'entrée.

On distingue deux types de complexités lorsqu'on aborde la question de la complexité de manière générale : la complexité temporelle et de la complexité spatiale, généralement, le temps de calcul est considéré comme la ressource la plus significative pour évaluer la complexité d'un algorithme. Dans ce manuscrit, lorsqu'il est fait mention de complexité et sauf mention contraire, il est admis que c'est de la complexité temporelle qu'il s'agit.

1.1.1 Généralité

De manière générale, soient un algorithme $\mathcal{A}$ pour résoudre un problème, et n la taille de la donnée d'entrée, on note $\mathcal{T}_{\mathcal{A}}(n)$, une fonction sur l'entier positif, le temps nécessaire à l'exécution de l'algorithme $\mathcal{A}$ pour la donnée de taille n. D'où le temps nécessaire est une définition précise comme le nombre d'opérations élémentaires qui ne dépend pas de la vitesse d'exécution de la machine ni de la qualité du code écrit par le programmeur.

Exemple 1.1. *Supposons que le problème posé soit de trouver un nom dans un annuaire du personnel qui consiste en une liste de taille n triée alphabétiquement. Pour résoudre ce problème, il existe plusieurs méthodes différentes. Considérons l'approche la plus naïve, que nous nommons $\mathcal{A}$, qui consiste à parcourir l'annuaire dans l'ordre à partir du premier nom jusqu'à trouver le nom recherché. Pour cette méthode, plusieurs cas peuvent arriver :*

- *Le meilleur cas arrive si le nom recherché est juste le premier nom dans l'annuaire, dans ce cas,* $\mathcal{T}_{\mathcal{A}}(n) = 1$
- *Le pire cas arrive si le nom recherché est situé au dernier rang dans l'annuaire, dans ce cas,* $\mathcal{T}_{\mathcal{A}}(n) = n$
- *Le cas moyen dépend de la répartition probabiliste des éléments dans l'annuaire. Dans le cas d'une distribution uniforme, le nombre moyen d'opérations pour trouver le nom recherché est* $\mathcal{T}_{\mathcal{A}}(n) = \frac{n}{2}$.

Définition 1.1 (complexité algorithmique)**.** *Soit* n *la taille de la donnée en entrée,* $f(n)$ *une fonction définie sur l'entier positif. Un algorithme* $\mathcal{A}$ *est de complexité* $\mathcal{O}(f(n))$ *dans le pire des cas s'il existe certaines constantes* c *et* N_0 *telles que :*

$$\forall n > N_0; \mathcal{T}_{\mathcal{A}}(n) < c \times f(n)$$

C'est-à-dire $\mathcal{T}_{\mathcal{A}}(n)$ croît au plus aussi vite que $f(n)$.

Exemple 1.2. *L'algorithme utilisé dans l'exemple 1.1 est de complexité* $\mathcal{O}(n)$.

Définition 1.2 (algorithme polynomial)**.** *Un algorithme* $\mathcal{A}$ *exécuté sur une entrée de taille* n *est dit polynomial s'il existe un entier* i *tel que* $\mathcal{A}$ *est de complexité* $\mathcal{O}(n^i)$.

Exemple 1.3. *Supposons que le nombre d'opérations nécessaire à la terminaison d'un algorithme* $\mathcal{A}$ *sur une entrée de taille* n *est exactement* $2n^2 - n - 7$*, alors* $\mathcal{A}$ *est un algorithme polynomial. Au lieu de dire* $\mathcal{A}$ *est en* $\mathcal{O}(n^2)$*, nous écrivons communément* $\mathcal{T}_{\mathcal{A}}(n) = \mathcal{O}(n^2)$.

Notons bien que la notion de complexité est utilisée pour spécifier une borne supérieure sur la croissance de la fonction. Elle n'est pas donnée de manière exacte. L'observation importante est que si la fonction du temps d'exécution est quadratique, alors si on double la taille d'entrée, le temps d'exécution va augmenter à quatre fois du temps actuel, et ceci ne dépend pas de la vitesse d'exécution de la machine.

Définition 1.3 (algorithme exponentiel)**.** *Un algorithme* $\mathcal{A}$ *exécuté sur une entrée de taille* n *est dit exponentiel s'il existe un réel* r *strictement supérieur à* 1 *et un polynôme* $f(n)$ *en n tel que* $\mathcal{A}$ *est de complexité de* $\mathcal{O}(r^{f(n)})$.

En pratique, il y a évidement une différence de temps de résolution entre les algorithmes polynômiaux et les algorithmes exponentiels. Pour donner un ordre d'idée sur les différentes complexités, le tableau ci-dessus présente les différents types de complexités et leur temps d'exécution.

Les différences de temps nécessaires à la résolution de problèmes avec des complexités différentes peuvent être vues clairement dans le tableau 1.1, elles peuvent

Complexité	Type de Complexité	Temps pour $n = 10$	Temps pour $n = 250$	Temps pour $n = 1000$
$\mathcal{O}(1)$	constante	$10ns$	$10ns$	$10ns$
$\mathcal{O}(log(n))$	logarithmique	$10ns$	$20ns$	$30ns$
$\mathcal{O}(\sqrt{n})$	racinaire	$32ns$	$158ns$	$316ns$
$\mathcal{O}(n)$	linéaire	$100ns$	$2.5\mu s$	$10\mu s$
$\mathcal{O}(nlog(n))$	linéarithmique	$100ns$	$6\mu s$	$30\mu s$
$\mathcal{O}(n^2)$	quadratique (polynomial)	$1\mu s$	$625\mu s$	$10ms$
$\mathcal{O}(n^3)$	cubique (polynomial)	$10\mu s$	$156ms$	$10s$
$\mathcal{O}(e^n)$	exponentielle	$10\mu s$	$10^{59}ans$	$> 10^{100}ans$
$\mathcal{O}(n!)$	factorielle	$36ms$	$> 10^{100}ans$	$> 10^{100}ans$

TABLE 1.1 – Évolution du temps de calcul en fonction de la complexité d'un algorithme et de la taille des données. Les temps d'exécutions sont estimés sur la base d'un accès mémoire de $10ns$ par étape.

être phénoménales. Plus précisément, les problèmes d'une complexité exponentielle ou factorielle sont impossibles à résoudre sur l'ensemble des données de taille raisonnable (n>250). Compte tenu de cette particularité, deux questions s'imposent :

- existe-t-il des problèmes qui n'admettent pas d'algorithme polynomial ?
- si oui, est-il possible de déterminer qu'un problème n'admet pas d'algorithme polynomial ?

Ce sont à ces questions que tente de répondre la théorie de la complexité présentée dans la section suivante.

1.1.2 La machine de Turing

Les machines de Turing ne sont pas des machines matérielles, elles sont une abstraction des ordinateurs. Une machine de Turing se compose d'une partie de contrôle et d'une bande infinie sur laquelle se trouvent écrits des symboles. La partie de contrôle est constituée d'un nombre fini d'états possibles et de transitions qui régissent des calculs de la machine. Les symboles de la bande sont lus et écrits par l'intermédiaire d'une tête de lecture. La partie de contrôle représente le microprocesseur. Un élément essentiel est que le nombre d'états est fini. Ceci prend en compte que les microprocesseurs possèdent un nombre déterminé de registres d'une taille fixe et que le nombre de configuration possible est fini. La bande représente la mémoire de l'ordinateur. Ceci comprend la mémoire centrale ainsi que les mémoires externes telles les disques durs. La tête de lecture représente le bus qui relie le microprocesseur à la mémoire. Contrairement à un ordinateur, la mémoire d'une machine de Turing est infinie. Ceci prend en compte qu'on peut ajouter des disques durs à un ordinateur de façon (presque) illimitée. Une autre différence entre une machine de Turing et un ordinateur est que l'ordinateur peut accéder à la mémoire de

manière directe (appelée aussi aléatoire) alors que la tête de lecture de la machine de Turing est décrite par les transitions de la machine.

Nous donnons maintenant la définition précise d'une machine de Turing :

Définition 1.4 (la machine de Turing)**.** *La mise en œuvre concrète d'une machine de Turing est réalisée par les éléments suivants :*

1. *Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini : $"\Gamma"$. L'alphabet contient un symbole spécial « blanc » : $"\#"$, et un ou plusieurs autres symboles. Le ruban est supposé être de longueur infinie vers la gauche ou vers la droite, en d'autres termes la machine doit toujours avoir assez de longueur de ruban pour son exécution. Les cases non encore écrites du ruban contiennent le symbole « blanc » ;*
2. *Une « tête de lecture/écriture » qui permet de lire et d'écrire les symboles sur le ruban, et de se déplacer vers la gauche ou vers la droite du ruban ;*
3. *Un « registre d'état » $"q"$ qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini et il existe un état spécial appelé « état de départ » qui est l'état initial de la machine avant son exécution ;*
4. *Une « fonction de transitions » $"t"$ qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture ($"G"$ pour gauche, $"D"$ pour droite), et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête.*

De manière formelle,

Une machine de Turing $\mathcal{M}$ est un septuplet$(Q, \Sigma, \Gamma, E, q_0, F, \#)$ où :
- *Q est l'ensemble des états de contrôle. C'est un ensemble fini$\{q_0, ..., q_n\}$;*
- *Σ est l'alphabet d'entrée. C'est un ensemble fini de symboles qui ne contient pas le symbole blanc : $\#$. Cet alphabet est utilisé pour écrire la donnée initiale sur la bande ;*
- *Γ est l'alphabet de la bande. C'est un ensemble fini qui comprend tous les symboles qui peuvent être écrits sur la bande. Ceci inclut bien sûr l'alphabet d'entrée Σ et le symbole blanc $\#$;*
- *E est un ensemble fini de transition de la forme (p, a, q, b, x) où p et q sont des états, a et b sont des symboles de bande et x est un élément de G,D. Une transition (p, a, q, b, x) est aussi notée $(p, a \to q, b, x)$;*
- *q_0 est l'état initial. C'est un état spécial de Q dans lequel se trouve machine au début d'un calcul ;*
- *F est l'ensemble des états finaux appelés aussi état d'acceptation ;*

– $\#$ *est le symbole blanc qui, au départ, remplit toutes les positions de la bande autres que celles contenant la donnée initiale.*

L'ensemble E de transition est aussi appelé fonction de transition et est notée δ.

Définition 1.5 (machine de Turing déterministe)**.** *Une machine de Turing $\mathcal{M}$: $(Q, \Sigma, \Gamma, E, q_0, F, \#)$ est déterministe si pour tout état p dans Q et tout symbole a dans Γ, il existe au plus une transition de la forme $(p, a \to q, b, x)$.*

Définition 1.6 (machine de Turing non-déterministe)**.** *La seule différence entre une machine de Turing déterministe et une machine de Turing non-déterministe porte sur la fonction de transition. Lorsque la machine est déterministe, la fonction δ associe à chaque paire (p, a) l'unique triplet(q, b, x), s'il existe, tel que $(p, a \to q, b, x)$ soit une transition. Lors que la machine est non-déterministe, la fonction δ associe à chaque paire (p, a) l'ensemble des triplets (q, b, x) tels que $(p, a \to q, b, x)$ soit une transition.*

En dépit de la contradiction apparente des termes, une machine de Turing déterministe peut être considéré comme un cas particulier de machine de Turing non-déterministe : c'est le cas pour lequel l'ensemble des triplets (q, b, x) possible est un singleton.

Bien que d'apparence très frustres, les machines de Turing fournissent un modèle universel pour les ordinateurs et les algorithmes. En fait, elles permettent de calculer toutes les fonctions récursives [1] et la « thèse de Church » (de ce fait présentée parfois comme la « thèse de Church-Turing ») peut se reformuler, en termes non techniques de la manière suivante : tout ce qui est calculable l'est avec une machine de Turing. Plus précisément, il est possible de montrer que tout calcul nécessitant un temps $f(n)$ sur une machine à accès aléatoire [2] peut être traduit sur une machine de Turing effectuant le même calcul en temps $\mathcal{O}(f(n)^6)$, voire même $\mathcal{O}(f(n)^3)$ avec une machine à plusieurs rubans (Papadimitriou 1994).

Malgré cette augmentation non négligeable du temps nécessaire à l'exécution d'un programme, cette propriété permet l'utilisation de la machine de Turing comme référence dans les définitions théoriques des classes de complexité.

1. Les fonctions récursives désignent la classe des fonctions calculables, autrement dit les fonctions dont les valeurs peuvent être calculées à partir de leurs paramètres par un processus mécanique.

2. Les ordinateurs actuels sont conçus d'après un modèle de machine à accès aléatoire. Ces machines possédant une mémoire adressable ont un accès direct à l'information, contrairement à la machine de Turing qui est obligée de parcourir séquentiellement son ruban pour obtenir un résultat intermédiaire.

1.1.3 Classes de complexité des problèmes de décision

La machine de Turing peut être utilisée pour définir des classes de problèmes selon leurs difficultés intrinsèques, ce que l'on appelle complexité des problèmes (non des algorithmes).

La théorie de la complexité vise à savoir si la réponse à un problème peut être donnée très efficacement ou au contraire être inatteignable en pratique et en théorie, avec des niveaux intermédiaires de difficulté entre les deux extrêmes ; pour cela, elle se fonde sur une estimation théorique des temps de calcul et de besoin de mémoire informatique. Dans le but de mieux comprendre comment les problèmes se placent les uns par rapport aux autres, la théorie de la complexité établit des hiérarchies de difficultés entre les problèmes algorithmiques, dont les niveaux sont appelés des « classes de complexité ».

Dans ce manuscrit, comme indiqué précédemment, nous ne nous intéressons qu'à la complexité temporelle.

Classes de complexité

Définition 1.7 (classe P)**.** *La classe P (pour polynomial) regroupe l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing déterministe.*

Par définition, la classe P contient les problèmes de décision polynomiaux : les problèmes qu'on peut résoudre à l'aide d'un algorithme déterministe en temps polynomial par rapport à la taille de l'instance. Nous allons définir similairement une classe plus vaste :

Définition 1.8 (classe NP)**.** *La classe NP regroupe l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing non-déterministe.*

Définition 1.9 (classe $CoNP$)**.** *La classe $CoNP$ regroupe l'ensemble des problèmes de décision dont les problèmes complémentaires appartiennent à la classe NP.*

Comme indiqué précédemment, une machine de Turing déterministe est une machine de Turing non-déterministe particulière, nous obtenons tout de suite la propriété suivante :

Propriété 1.1. *Nous avons : $P \subseteq NPetP \subseteq CoNP$.*

Définition 1.10 (C-difficile et C-complet)**.** *Soit C une classe de complexité (comme P, NP, etc.). On dit qu'un problème est C-difficile si ce problème est au moins aussi difficile que tous les problèmes dans C. Un problème C-difficile qui appartient à C est dit C-complet.*

Afin de définir une véritable classification des problèmes, la notion de réduction polynomial est utilisée. En fait, la notion de réduction permet de « savoir » si un problème est aussi difficile qu'un autre.

Définition 1.11 (réduction polynomial)**.** *Soient Π et Π' deux problèmes de décision, une réduction de Π à Π' est un algorithme polynomial transformant toute instance de Π' en une instance de Π. Ainsi, si l'on a un algorithme pour résoudre Π, on sait aussi résoudre Π'*

Remarque 1.1. *Quand on parle de problème C-difficile ou C-complet, on s'autorise uniquement des réductions dans* LOGSPACE.

La relation de réduction étant réflexive et transitive, elle définit un préordre sur les problèmes.

On a NP-complet $\subseteq$ NP-difficile, par contre un problème NP-difficile n'est pas nécessairement dans NP.

Si un problème NP-complet est dans la classe P (NP-complet $\cap P \neq \emptyset$), alors $P = NP$. Mais malheureusement ou heureusement, personne n'a trouvé d'algorithme polynomial pour un problème NP-complet. En outre, personne n'a pu prouver qu'il n'existe pas.

Elle constitue l'une des questions ouvertes fondamentales de la théorie de la complexité.

Les problèmes sont classés de façon incrémentale, la classe d'un nouveau problème étant déduite d'un ancien problème. Il a toutefois été nécessaire de définir un « premier » problème NP-complet afin de classer tous les autres. Ce premier problème à avoir été démontré NP-complet, par Cook (1971), est le problème de la satisfaisabilité propositionnelles (SAT).

1.2 Le problème SAT

Le problème de la satisfiabilité d'une formule propositionnel ou « problème SAT » est un problème de décision visant à savoir s'il existe une valuation sur un ensemble de variables propositionnelles telle qu'une formule propositionnelle donnée soit logiquement `vrai` . La résolution du problème SAT est très importante en

théorie de la complexité (il représente le problème NP-complet de référence Cook (1971)) et a de nombreuses applications en *planification classique, vérification formelle, bioinformatique,* et jusqu'à la configuration d'un ordinateur ou d'un *système d'exploitation,* etc. Le problème SAT étant un problème de logique propositionnelle, nous commençons par introduire quelques notions de la logique propositionnelle afin de permettre une meilleure comprehension de ce dernier.

1.2.1 Logique propositionnelle

Syntaxe

Nous abordons ce sujet par la définition des principaux éléments syntaxiques de la logique propositionnelle. Pour spécifier la syntaxe, il est nécessaire de définir d'abord les atomes pouvant être utilisés pour formuler des énoncés :

Définition 1.12 (atome)**.** *Une proposition atomique (ou atome) est une variable booléenne prenant ses valeurs dans l'ensemble* `faux` ,`vrai` *ou 0,1 ou* $\bot$,$\top$.

Définition 1.13 (langage propositionnel)**.** *Soit* $\mathrm{P}rop$ *un ensemble fini de symboles propositionnels appelés également variables ou atomes. Pour chaque sous-ensemble* $\mathcal{V}$ *de* $\mathrm{P}rop$, $\mathrm{P}rop_{\mathcal{V}}$, *désigne le langage propositionnel construit à partir des symboles de* $\mathcal{V}$, *des parenthèses* "(" *et* ")", *des constantes booléennes* `faux` *(*$\bot$*) et* `vrai` *(*$\top$*), et des connecteurs logiques :* $\neg$ *pour la négation,* $\wedge$ *est utilisé pour la conjonction,* $\vee$ *pour la disjonction,* $\Rightarrow$ *pour l'implication,* $\Leftrightarrow$ *pour l'équivalence et* $\oplus$ *pour le ou exclusif. Une suite finie d'éléments du langage est une expression.*

Définition 1.14 (formule propositionnelle)**.** *L'ensemble des formules propositionnelles est le plus petit ensemble d'expressions tel que :*

- *les atomes* $\bot$ *et* $\top$ *sont des formules ;*
- *si* $\mathcal{F}$ *est une formule alors* $\neg\mathcal{F}$ *est une formule ;*
- *si* $\mathcal{F}$ *et* $\mathcal{F}'$ *sont des formules alors :*
  - $(\mathcal{F} \wedge \mathcal{F}')$ *est une formule ;*
  - $(\mathcal{F} \vee \mathcal{F}')$ *est une formule ;*
  - $(\mathcal{F} \Rightarrow \mathcal{F}')$ *est une formule ;*
  - $(\mathcal{F} \Leftrightarrow \mathcal{F}')$ *est une formule ;*

Définition 1.15 (littéral)**.** *Un littéral est une variable propositionnelle* x *(littéral positif) ou sa négation* $\neg x$ *(littéral négatif). Les deux littéraux* x *et* $\neg x$ *sont dits complémentaires. On note également* $\neg l$ *ou* $\bar{l}$ *le littéral complémentaire de* l.

Définition 1.16 (littéral pur (monotone))**.** *Un littéral* l *est dit pur (monotone) pour une formule* $\mathcal{F}$ *si et seulement si* l *apparaît dans* $\mathcal{F}$ *et* $\neg l$ *n'apparaît pas dans* $\mathcal{F}$.

Nous fixons ici quelques notations supplémentaires usuelles nécessaires pour la suite. Si l est un littéral alors $|l|$ désigne la variable correspondante. Pour un ensemble de littéraux $\mathcal{L}$, $\bar{\mathcal{L}}$ désigne l'ensemble $\bar{l}|l \in \mathcal{L}$. Étant donnée une formule propositionnelle $\mathcal{F}$ appartenant à $\mathrm{P}rop_{\mathcal{V}}$, on note $\mathcal{V}(\mathcal{F})$ l'ensemble des variables propositionnelles apparaissant dans $\mathcal{F}$ et $\mathcal{L}(\mathcal{F})$ l'ensemble des littéraux apparaissant dans $\mathcal{F}$. $\mathcal{L}(\mathcal{F})$ est dit complet ssi pour tout $l \in \mathcal{L}(\mathcal{F})$, on a $\bar{l} \in \mathcal{L}(\mathcal{F})$.

Sémantique

Définition 1.17 (interprétation)**.** *Une interprétation $\mathcal{I}$ en calcul propositionnel est une application associant à toute formule propositionnelle $\mathcal{F}$ une valeur $\mathcal{I}(\mathcal{F})$ dans* `faux` , `vrai` *. L'interprétation $\mathcal{I}(\mathcal{F})$ d'une formule $\mathcal{F}$ est définie par la valeur de vérité donnée à chacun des atomes de $\mathcal{F}$. $\mathcal{I}(\mathcal{F})$ se calcule par l'intermédiaire de règles suivantes :*

1. $\mathcal{I}(\top)$ = `vrai` *;*
2. $\mathcal{I}(\bot)$ = `faux` *;*
3. $\mathcal{I}(\neg\mathcal{F})$ = `vrai` *ssi* $\mathcal{I}(\mathcal{F})$ = `faux` *;*
4. $\mathcal{I}(\mathcal{F} \wedge \mathcal{G})$ = `vrai` *ssi* $\mathcal{I}(\mathcal{F}) = \mathcal{I}(\mathcal{G})$ = `vrai` *;*
5. $\mathcal{I}(\mathcal{F} \vee \mathcal{G})$ = `faux` *ssi* $\mathcal{I}(\mathcal{F}) = \mathcal{I}(\mathcal{G})$ = `faux` *;*
6. $\mathcal{I}(\mathcal{F} \Rightarrow \mathcal{G})$ = `faux` *ssi* $\mathcal{I}(F)$ = `vrai` *et* $\mathcal{I}(\mathcal{G})$ = `faux` *;*
7. $\mathcal{I}(\mathcal{F} \Leftrightarrow \mathcal{G})$ = `vrai` *ssi* $\mathcal{I}(\mathcal{F}) = \mathcal{I}(\mathcal{G})$ *;*

L'interprétation $\mathcal{I}$ d'une formule $\mathcal{F}$ est dite complète ssi $\forall x \in \mathcal{V}(\mathcal{F})$, on a soit $\mathcal{I}(x) =$ `vrai` , soit $\mathcal{I}(x) =$ `faux` ; elle est dite partielle sinon.

Remarque 1.2. *Dans la suite de ce manuscrit, lorsqu'aucune information n'est apportée sur la nature de l'interprétation, elle est considérée comme complète.*

On représente souvent une interprétation $\mathcal{I}$ comme un ensemble de littéraux, *i.e.*, $x \in \mathcal{I}$ si $\mathcal{I}(x) =$ `vrai` , $\neg x \in \mathcal{I}$ si $\mathcal{I}(x) =$ `faux` . On note $\mathcal{S}(\mathcal{F})$ l'ensemble des interprétations d'une formule $\mathcal{F}$. Si $\mathcal{F}$ possède exactement n variables propositionnelles alors $|\mathcal{S}(\mathcal{F})| = 2^n$. Nous définissons la distance entre deux interprétations $\mathcal{I}$ et $\mathcal{I}'$ de $\mathcal{S}$ comme suit :

Définition 1.18 (distance de hamming entre deux interprétations)**.** *Soient $\mathcal{I}$ et $\mathcal{I}'$ deux interprétations d'une formule propositionnelle $\mathcal{F}$. La distance entre $\mathcal{I}$ et $\mathcal{I}'$ est la distance de Hamming entre eux, notée $Dis_{\mathfrak{h}}(\mathcal{I}, \mathcal{I}')$, est définie par $|E_{\mathfrak{h}}(\mathcal{I}, \mathcal{I}')|$ tel que $E_{\mathfrak{h}}(\mathcal{I}, \mathcal{I}') = \{x \in \mathcal{V}_{\mathcal{F}}$ telle que $\mathcal{I}(x) \neq \mathcal{I}'(x)\}$.*

Exemple 1.4. *Soient* $\mathcal{F} = a \vee (b \wedge c)$, $\mathcal{I} = \{a, b, c\}$ *et* $\mathcal{I}' = \{\neg a, \neg b, c\}$ *;* $E_{\mathfrak{h}}(\mathcal{I}, \mathcal{I}') = \{a, b\}$ *et* $Dis_{\mathfrak{h}}(I, \mathcal{I}') = 2$.

Définition 1.19 (modèle)**.** *Une interprétation* $\mathcal{I}$ *satisfait une formule* $\mathcal{F}$ *si* $\mathcal{I}(\mathcal{F})$ = `vrai` *. On dit alors que* $\mathcal{I}$ *est un modèle de* $\mathcal{F}$*. Inversement, si* $\mathcal{I}(\mathcal{F})$ = `faux` *, on dit que* $\mathcal{I}$ *falsifie* $\mathcal{F}$ *et que* $\mathcal{I}$ *est un contre-modèle de* $\mathcal{F}$ *(ou nogood), et qu'on note communément* $\mathcal{I} \models \neg\mathcal{F}$.

On note $M(\mathcal{F})$ l'ensemble des modèles d'une formule $\mathcal{F}$.

Définition 1.20 (impliquant premier)**.** *Soit* $\mathcal{F}$ *une formule* CNF *et* $\mathcal{I}$ *un modèle de* $\mathcal{F}$*,* $\mathcal{I}$ *est appelé impliquant premier de* $\mathcal{F}$ *si pour tout* $x \in \mathcal{I}$*,* $\mathcal{I} \setminus x$ *n'est pas un modèle de* $\mathcal{F}$.

Définition 1.21 (formule satisfiable)**.** *Une formule propositionnelle* $\mathcal{F}$ *est satisfiable si elle admet au moins un modèle, i.e.,* $M(\mathcal{F}) \neq \emptyset$.

Définition 1.22 (formule insatisfiable)**.** *Une formule propositionnelle* $\mathcal{F}$ *est dite insatisfiable s'il elle n'admet pas de modèle, i.e.,* $M(\mathcal{F}) = \emptyset$.

Définition 1.23 (tautologie)**.** *Une formule propositionnelle* $\mathcal{F}$ *est une tautologie si toute interprétation de* $\mathcal{F}$ *est un modèle de* $\mathcal{F}$*, i.e.,* $M(\mathcal{F}) = \mathcal{S}(\mathcal{F})$

Définition 1.24 (conséquence logique)**.** *Soient* $\mathcal{F}$*,* $\mathcal{G}$ *deux formules propositionnelles, on dit que* $\mathcal{G}$ *est conséquence logique de* $\mathcal{F}$*, noté* $\mathcal{F} \models \mathcal{G}$*, ssi* $M(\mathcal{F}) \subseteq M(\mathcal{G})$*.* $\mathcal{F}$ *et* $\mathcal{G}$ *sont équivalentes ssi* $\mathcal{F} \models \mathcal{G}$ *et* $\mathcal{G} \models \mathcal{F}$.

Remarque 1.3. *De la définition d'une interprétation on déduit les équivalences suivantes :* $(\mathcal{F} \Rightarrow \mathcal{G}) \equiv \neg\mathcal{F} \vee \mathcal{G}$ *et* $(\mathcal{F} \Leftrightarrow \mathcal{G}) \equiv (\mathcal{F} \Rightarrow \mathcal{G}) \wedge (\mathcal{G} \Rightarrow \mathcal{F})$.

A ce niveau, nous introduisons un théorème très important qui nous permet d'énoncer un résultat fondamentale en démonstration automatique (preuve par l'absurde).

Théorème 1.2 (déduction)**.** *Soient* $\mathcal{F}$ *et* $\mathcal{G}$ *deux formules propositionnelles,* $\mathcal{F} \models \mathcal{G}$ *ssi* $\mathcal{F} \wedge \neg\mathcal{G}$ *est une formule insatisfiable.*

Ce théorème indique que prouver l'implication sémantique est équivalent à montrer l'inconsistance d'une formule. Il est exploité par la plupart des algorithmes de démonstration automatique autour du problème SAT.

Formes normales

Nous nous intéressons dans cette section aux différentes formes normales. De ce fait, il est intéressant d'introduire tout d'abord la notion de clause :

Définition 1.25 (clause)**.** *Une clause est une disjonction finie de littéraux. Une clause ne contenant pas de littéraux complémentaires est dite fondamentale, sinon elle est dite tautologique.*

Définition 1.26 (différentes variantes de clauses)**.** *Soit c une clause , elle est dite :*

- (UNITAIRE) *ssi elle contient uniquement un seul littéral.*
- (BINAIRE) *binaire ssi elle contient exactement deux littéraux.*
- (POSITIVE, NÉGATIVE, MIXTE) *positive (resp. négative) si elle ne contient que des littéraux positifs (resp. négatifs). Une clause constituée de littéraux positifs et négatifs est appelée clause mixte.*
- (VIDE) *La clause vide, notée $\perp$, est une clause ne contenant aucun littéral. Elle est par définition insatisfiable.*
- (CLAUSE TAUTOLOGIQUE) *elle contient un littéral et son complémentaire. Elle est par définition vraie.*
- (CLAUSE HORN, CLAUSE REVERSE-HORN) *Une clause Horn (resp. reverse-Horn) est une clause qui contient au plus un littéral positif (resp. négatif).*

Définition 1.27 (Différents types d'instances)**.** *Soit $\mathcal{F}$ une instance* SAT*, $\mathcal{F}$ est une instance :*

- *k-*SAT *si toutes ses clauses contiennent exactement $k \in \mathbb{N}$ littéraux. Hormis* 1-SAT *et* 2-SAT *Cook (1971), le problème k-*SAT *est généralement NP-complet ;*

- *Horn-*SAT *(respectivement Reverse-Horn-*SAT*) si toutes ses clauses sont Horn (respectivement reverse-horn). Il a été démontré dans la littérature que le test de satisfiabilité d'un ensemble de clauses de Horn ou reverse-horn se fait en temps polynomiale, mieux encore en temps linéaire Minoux (1988), Dalal (1992), Rauzy (1995) ;*

Définition 1.28 (terme)**.** *Un terme est une conjonction finie de littéraux.*

Définition 1.29 (formes normales)**.** *Nous distinguons deux formes normales particulières pour les propositions :*

- (FORME CNF) *Une formule $\mathcal{F}$ est sous* FORME NORMALE CONJONCTIVE (CNF) *si et seulement si $\mathcal{F}$ est une conjonction de clauses.*
- (FORME DNF) *Une formule $\mathcal{F}$ est sous* FORME NORMALE DISJONCTIVE (DNF) *si et seulement si $\mathcal{F}$ est une disjonction de termes.*

Exemple 1.5. *Les formules $[(a \vee b) \wedge ((\neg c) \vee d)]$ et $[(a \wedge b) \vee ((\neg c) \wedge d)]$ sont respectivement sous forme normale conjonctive et disjonctive.*

Dans la suite de ce manuscrit, lorsqu'il sera fait référence à une formule propositionnelle, sauf mention contraire, il s'agira d'une formule sous forme CNF.

Propriété 1.3. *Toute formule de la logique propositionnelle peut être réécrite sous une forme normale.*

Une approche permettant de transformer en temps et espace linéaire toute formule booléenne sous forme quelconque en une formule sous forme normale conjonctive équivalente du point de vue de la satisfiabilité est proposée par Tseitin (1968).

1.2.2 le problème SAT

Toutes les notions introduites dans les sections précédentes nous permettent de donner à présent la définition concrète du problème SAT ainsi que quelques règles fondamentales permettant de simplifier une formule CNF.

Nous commençons tout d'abord par introduire le problème de décision :

Définition 1.30 (Problème de décision)**.** *Un problème* Π *de décision est une question mathématiquement définie portant sur des paramètres données sous forme manipulable informatiquement et demandant une réponse : « Oui » ou « Non ». C'est-à-dire que l'ensemble* D_Π *des instances de* Π *peut être scindé en deux ensembles disjoints :*

1. Y_Π *: l'ensemble des instances telles qu'il existe un programme les résolvant et répondant « Oui » ;*
2. N_Π *: l'ensemble des instances pour lesquelles la réponse est « Non ».*

Définition 1.31 (problème de décision complémentaire)**.** *Soit* Π *un problème de décision, le problème complémentaire* Π^c *de* Π *est aussi un problème de décision tel que :*

$$D_\Pi = D_\Pi^c \text{ et } Y_\Pi = N_\Pi^c$$

De nombreux problèmes informatiques peuvent être réduits à des problèmes de décision. Un problème dont la réponse n'est ni « Oui » ni « Non » peut être simplement transformé en un problème de décision : « Existe-t-il une solution au problème ? »

Définition 1.32 (problème SAT)**.** *Le problème* SAT *est un problème de décision qui consiste à décider si une formule sous forme normale conjonctive (*CNF*) admet ou non un modèle.*

Exemple 1.6. *Soit une formule booléenne mise sous forme* CNF $\mathcal{F} = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a)$*, à la question, $\mathcal{F}$ admet-elle un modèle ? La réponse est oui : l'interprétation $\mathcal{I} = a, b, c$ satisfait la formule $\mathcal{F}$. En plus, $\mathcal{I}$ est le seul modèle de $\mathcal{F}$, donc la nouvelle formule $\mathcal{F} \wedge (\neg a \vee \neg b \vee \neg c)$ est insatisfiable.*

Plusieurs techniques peuvent être appliquées sur les clauses d'une formule CNF sans pour autant en changer sa satisfaisabilité. Généralement, elles visent soit à ajouter ou supprimer de la redondance dans les formules CNF.

La première approche présentée ici qui est la règle de résolution, est sans aucun doute l'une des règles les plus fondamentales de la logique propositionnelle.

Définition 1.33 (résolution)**.** *Soient deux clauses $c_i = (x \vee \alpha_i)$ et $c_j = ((\neg x) \vee \alpha_j)$ contenant respectivement x et $\neg x$, une nouvelle clause $r = \alpha_i \vee \alpha_j$ peut être obtenue par la suppression de toutes les occurrences du littéral x et $\neg x$ dans c_i et c_j. Cette opération, notée $\eta[x, c_i, c_j]$, est appelée résolution et la clause produite r est appelée résolvante.*

Exemple 1.7. *Soient $c_1 = (a \vee b \vee c)$ et $c_2 = (\neg a \vee c \vee d)$ deux clauses, nous avons $\eta[a, c_1, c_2] = (b \vee c \vee d)$.*

Propriété 1.4. *Soient $\mathcal{F}$ une formule, r la résolvante de c_i et c_j deux clauses de $\mathcal{F}$ contenant respectivement x et $\neg x$. Les deux propriétés suivantes sont vraies :*

- $c_i \wedge c_j \models r$
- $\mathcal{F} \equiv \mathcal{F} \wedge r$ *(de point de vu satisfaisabilité)*

Une autre règle importante est la règle de ***fusion*** qui consiste à remplacer les multiples occurrences d'un littéral par une seule occurrence de ce littéral.

Définition 1.34 (fusion)**.** *Soit $c_i = \{x_1, x_2, , ...x_n, l, y_1, y_2, ..., y_m, l, z_1, z_2, ..., z_k, l\}$, une clause dans une formule $\mathcal{F}$, c_i peut être remplacée par*
$c'_i = \{x_1, x_2, , ...x_n, y_1, y_2, ..., y_m, z_1, z_2, ..., z_k, l\}$. Cette opération est appelée fusion.

Les deux règles ci-dessus sont des règles permettant d'éliminer les littéraux dans les clauses. Nous introduisons à présent deux techniques permettant de supprimer les clauses dans une formule.

Définition 1.35 (subsumption)**.** *Une clause c_i subsume une autre clause c_j ssi $c_i \subseteq c_j$. Dans ce cas, $c_i \models c_j$*

Exemple 1.8. *Soient $c_1 = (x_1 \vee x_2)$ et $c_2 = (x_1 \vee x_2 \vee x_3)$ deux clauses, la clause c_1 subsume la clause c_2.*

Définition 1.36 (auto-subsumption)**.** *Une clause c_i auto-subsume une autre clause c_j ssi la résolvante de c_i et c_j subsume c_j*

Exemple 1.9. *Soient $c_1 = (x_1 \vee x_2)$ et $c_2 = (\neg x_1 \vee x_2 \vee x_3)$ deux clauses, la résolvante de c_1 et c_2 : $r = (x_2 \vee x_3)$ subsume la clause c_2, alors c_2 est auto-subsumée par c_1.*

Définition 1.37 (clause redondante)**.** *Soit c une clause dans une formule* CNF *$\mathcal{F}$, c est dite redondante ssi $\mathcal{F} \setminus c \models c$.*

L'application de ces techniques (subsumption et auto-subsumption) permettent de détecter des clauses redondantes, et la suppression de clauses redondantes permet de simplifier la formule en préservant sa satisfiabilité.

1.3 Problèmes autour de SAT

Le nombre de problèmes qui gravitent autour du problème SAT accentue l'importance de ce problème et élargit ses champs d'applications. Même si ces problèmes sont généralement beaucoup plus difficiles que SAT, ils bénéficient régulièrement de transferts ou d'extensions de résultats obtenus dans le cadre SAT. Nous rappelons certains des problèmes qui suscitent un intérêt croissant ces dernières années.

Définition 1.38 (MaxSAT)**.** *MaxSAT est le problème d'optimisation associé à SAT. Il s'agit, pour une formule* CNF *donnée $\mathcal{F}$ de déterminer le nombre maximal MaxSAT($\mathcal{F}$) de clauses de $\mathcal{F}$ qui peuvent être satisfaites.*

Remarque 1.4. *Pour une formule* CNF *$\mathcal{F}$ satisfiable composée de n clauses, on a MaxSAT($\mathcal{F}$) = n.*

On peut noter aussi que même restreint aux formules 2-SAT, ou encore 2-SAT monotone, le problème MaxSAT reste un problème NP-Difficile. MaxSAT est un cas particulier du problème MaxSAT pondéré

("Weighted MaxSAT"). En effet, pour Weighted MaxSAT, un poids est associé à chaque clause, l'objectif est de maximiser la somme des poids des clauses satisfaites.

Définition 1.39 (#sat)**.** *Soit $\mathcal{F}$ une formule* CNF*. #SAT est le problème qui consiste à déterminer le nombre de modèles de $\mathcal{F}$.*

Le problème de dénombrement de solutions #SAT est un problème #P-Complet. Au-delà de l'intérêt pratique que revêt la connaissance du nombre de solutions d'une instance, la résolution du problème #SAT trouve des applications dans différents domaines de l'intelligence artificielle tels que le diagnostic, l'inférence dans les réseaux bayésiens, etc.

1.4 L'évaluation empirique

Le caractère NP-complet du problème SAT est toujours un verrou théorique majeur, les performances des différents algorithmes proposés afin de résoudre ce problème sont généralement évaluées sur un ensemble de formules ou d'instances SAT. Mais comparer les différents prouveurs entre eux est une tâche difficile. Il est très difficile de déterminer à l'avance les performances d'un algorithme sur un problème donné. Toutesfois, Pour évaluer les performances des différents algorithmes (ou prouveurs) et pour encourager la recherche d'algorithmes toujours plus performants, des compétitions ((SAT *Race*, SAT *Competition* et SAT *Challenge*) sont organisées. Les algorithmes sont comparés avec des ressources limitées (temps et mémoire) sur un large panel de problèmes SAT (benchmark ou instances SAT) de différents types(*aléatoire, élaboré, industrielle*) . Dans la suite de ce manuscrit, lorsque les expérimentations sont présentées, les instances sont issues de ces différentes compétitions.

1.4.1 Instances aléatoires

Historiquement, la motivation d'étudier les instances aléatoires est de mieux comprendre la dureté des instances « typiques ». Dans Franco et Paull (1983) les auteurs ont proposé l'analyse des instances aléatoires k-SAT.

Définition 1.40 (instance aléatoire k-SAT)**.** *Une instance aléatoire k-SAT est une formule SAT $\mathcal{F}_k(n, m)$, avec m clauses sur n variables. toutes ses clauses sont choisies aléatoirement parmi $2^k \binom{n}{k}$ clauses de taille k, où les clauses sont construites en tirant uniformément k littéraux distincts parmi l'ensemble des $2 \times n$ littéraux du problème.*

Les instances aléatoires sont très convenables pour étudier les propriétésdu p roblème SAT et pour évaluer les solveurs SAT.

Plusieurs générateurs d'instances aléatoires sont proposés Franco (2001), mais le générateur d'instances aléatoires k-SAT est le plus étudié.

Sur cette catégorie d'instances, le seuil de difficulté est donné en fonction du rapport $\frac{m}{n}$. En effet dans Selman *et al.* (1996), les auteurs ont montré que pour $k \geq 3$, il existe un intervalle du rapport entre le nombre de clauses et le nombre de variables, $r = \frac{m}{n}$, dans lequel c'est difficile à *décider* si une instance aléatoire de k-SAT est satisfiable ou pas. Nous pouvons alors prévoir la difficulté de telles instances.

Par exemple, pour $k = 3$, si $r < 4$, une assignation `vrai` peut être trouvée presque facilement pour tous les instances ; et pour $r > 4.5$, tous les instances sont quasiment insatisfiables. Pour $r \approx 4.25$, une affectation `vrai` peut être trouvée pour une moitié des instances, et autour de cette valeur, le coût de calcul pour trouver une affectation `vrai` , si celle existe, est maximisé.

1.4.2 Instances élaborés

Les instances élaborées sont souvent les instances académiques comme les problèmes de coloration de graphes, des pigeons et des n-reines. Ces instances sont souvent créées à la main en codant le problème initial comme un problème de satisfaction de contraintes SAT. Ces instances ont leurs propriétés mathématiques dont le degré de difficulté est difficile à prévoir.

1.4.3 Instances industrielles

Les instances industrielles sont issues de problèmes industriels tels que les problèmes de vérification de circuits intégrés Velev et Bryant (2003), Vérification formelle bornée Biere *et al.* (1999) et planification Kautz et Selman (1996). Ces instances n'ont pas réellement de caractéristiques permettant de prévoir leur degré de difficultés, mais elles ont souvent leurs structures intrinsèques.

Alors qu'il n'y a aucune relation entre variables (et clauses) dans une instance aléatoire, Il peut y avoir différentes relations entre variables (et clauses) dans une instance industrielle. Parmi ces relations, nous pouvons citer entre autre : Les *symétries* (deux variables sont symétriques si leur permutation dans la formule donne la même formule) et les *équivalences* (deux littéraux sont équivalents si leur valeur est la même dans tous les modèles). Ces informations structurelles souvent perdues lors de la transformation en CNF peuvent s'avérer utiles pour la résolution Rauzy *et al.* (1999), Kautz *et al.* (1997). Nous proposons au chapitre 4, une nouvelle technique qui extrait et exploite certaines structures cachées dans la CNF pour réduire la taille des clauses.

1.5 Conclusion

Ce chapitre qui s'acheve a fait l'objet de la présentation du problème de la satisfiabilité propositionnelle SAT. Ce problème revient à décider si une formule booléenne mise sous forme normale conjonctive est satisfiable ou non. Il est le premier problème NP-complet et est au cœur de la théorie de la complexité où il joue un rôle fondamental. De ce fait, pour une meilleure comprehension de ce problème, il était important pour nous d'introduire tout d'abord quelques notions autour de la théorie de la complexité.

Le problème SAT qui est considéré comme le "père" des problèmes combinatoires de décision existe également sous plusieurs variantes dont quelques unes ont été présentées ici. Pour étudier les propriétés de ce problème et par la même évaluer les performances des solveurs SAT, il est important de bien choisir l'ensemble d'instances qui servira de base de tests. Trois catégories d'instances ont été présentées dans ce chapitre : les instances aléatoires, les instances élaborés et les instances industrielles. Dans les chapitres qui suivent, nous présentons plusieurs paradigmes pour la résolution pratique du problème SAT.

Chapitre 2

Méthodes de résolution

Sommaire

DAns la littérature, il existe plusieurs algorithmes permettant de résoudre le problème SAT. Ces algorithmes reposent en générale sur des principes très variés : énumération Quine (1950), Davis *et al.* (1962), Jeroslow et Wang (1990), résolution Robinson (1965), Galil (1977), recherche local Selman *et al.* (1992), Li et Huang (2005), diagrammes binaires de décision Akers (1978), Bryant (1992), Uribe et Stickel (1994), algorithmes génétiques et évolutionnistes Hao et Dorne (1994), Gottlieb *et al.* (2002), Lardeux *et al.* (2006) etc.

Plusieurs recherches sur la satisfiabilité propositionnelle sont focalisées sur trois méthodes de résolution, à savoir : les approches incomplètes, les approches complètes et les approches hybrides.

La plupart des algorithmes incomplets sont fondés sur le principe de recherche locale, très utilisé en recherche opérationnelle. Le principe de base des méthodes de recherche locale consiste à se déplacer judicieusement dans l'espace des configurations en améliorant la configuration courante. Ils s'arrêtent un temps fixé (une constante) et donnent deux types de réponses : soit ils retournent 'oui' et indiquent que l'instance est satisfiable, ou ils retournent l'expression 'impossible de conclure'. Dans le cas des instances satisfiables, les configurations sont des interprétations

complètes de la CNF et l'amélioration se juge en terme de nombre de clauses falsifiées par les interprétations. Alors, les méthodes de recherche locale permettent de trouver des interprétations pas nécessairement modèles de la formule mais de bonne qualité dans des temps de calcul raisonnables. Un des inconvénients majeurs de ce type d'algoritmes est qu'ils sont limités à la recherche d'un modèle sans garantir son obtention (sauf adaptation particulière) même considérant un temps infini. Dans ce cas l'instance admet peut-être une solution mais l'algorithme est incapable de la fournir. La recherche locale pour SAT s'appuie sur un parcours non systématique de l'espace de recherche, c'est-à-dire de l'espace de toutes les interprétations possibles. A chaque pas seulement quelques interprétations sont examinées. Ils permettent d'explorer l'espace de recherche de manière plus flexible. En effet ces méthodes de recherche n'imposent pas un ordre sur l'examen des interprétations par rapport aux méthodes complètes basées sur la procédure DPLL, où les interprétations sont testées suivant un ordre induit par l'heuristique de branchement. De nombreuses techniques de recherche locale ont été appliquées à SAT avec plus ou moins de succès. Néanmoins, Ces techniques se sont montrées très efficaces pour la recherche des modèles des formules CNF sur les instances aléatoires.

Les algorithmes complets parcourent, en un temps fini tout l'arbre de recherche, et garantissent de trouver pour toute formule exprimée sous forme CNF une solution satisfiable si elle existe, sinon de prouver que la formule n'admet pas de solution. Ces algorithmes donnent toujours une réponse si on leur accorde assez de temps et d'espace car ils effectuent une recherche explicite et exhaustive dans l'espace des instanciations. Ils s'appuient généralement sur un parcours en profondeur d'un arbre de recherche, où chaque nœud correspond à l'affectation d'une variable, ce nœud représente une sous-formule obtenue à partir de la formule originale simplifiée par l'instanciation courante des variables et chaque chemin correspondant à une interprétation partielle des variables de la formule. Ceci permet de construire progressivement un modèle, variable par variable. L'objectif est donc de déterminer un chemin de la racine à une feuille. Ce chemin représente une interprétation complète des variables de la formule qui satisfait l'ensemble des contraintes du problème où de déterminer qu'il n'existe pas un tel chemin. Le nombre d'étapes nécessaires pour ce type d'algorithme est proportionnel au nombre d'instanciations possibles, et est donc d'ordre exponentiel en fonction du nombre de variables de la formule. Au pire des cas on doit essayer les 2^n instanciations possibles. Il est par conséquent quasiment impossible d'obtenir une réponse dans une durée raisonnable si la taille du problème devient trop grande. La complexité spatiale est linéaire pour ces algorithmes puisqu'on stocke uniquement les clauses ainsi que l'instanciation courante.

Les approches hybrides permettent de combiner les algorithmes de recherche locale et les approches complètes pour résoudre un même problème. L'idée ici est

de tirer profit des avantages offerts par chacun des algorithmes afin d'obtenir un algorithme au moins aussi efficace que chacun des approches prises séparement. le but de la combinaison est de produire sur chaque instance à résoudre un ordre d'exécution des algorithmes dans le temps et dans l'espace.

Dans ce manuscrit, nous nous attelons uniquement sur les approches complètes, lesquelles ont fait l'objet des travaux de cette thèse. Dans ce chapitre, nous rappelons tout d'abord la notion de systèmes de preuve, par la suite, après avoir introduit la résolution de Robinson (1965) et un ensemble d'approches s'appuyant sur ce principe, nous présentons une approche énumérative basée sur la notion d'arbre sémantique ainsi que les améliorations apportées à celle-ci et qui ont conduit à l'approche DPLL (algorithme de Quine (1950), simplification et heuristique de choix de variable). Ce chapitre se termine par la présentation des solveurs SAT modernes (CDCL) ainsi que des différents composants participant à leur efficacité (redémarrage, heuristique de choix de variable VSIDS et apprentissage).

2.1 Systèmes de Preuve

Un système de preuve (propositionel) (Cook (1971)) est un algorithme en temps polynomial SP tel que pour toute formule propositionelle insatisfaisable $\mathcal{F}$, il existe une preuve p telle que SP accepte l'entrée ($\mathcal{F}$, p), c'est à dire que SP vérifie que p est bien une preuve d'insatisfaisabilité pour $\mathcal{F}$. En d'autres termes, un système de preuve est un moyen efficace de vérifier la validité d'une preuve d'insatisfaisabilité. Mais trouver une telle preuve reste difficile dans le cas général. Les systèmes de preuves ont été étudiés en informatique théorique comme une façon de comparer les problèmes NP et co-NP. Ils permettent également de comparer les systèmes d'inférences les uns par rapport aux autres en permettant par exemple d'établir une hiérarchie de ces derniers reflétant leurs puissances relatives.

Un système de preuves SP admet des preuves courtes (resp. longues) pour une famille d'instances I si la taille de la plus petite preuve croît polynomialement (resp. exponentiellement) avec la taille des instances de I. On dit alors que I est simple (resp. difficile) pour SP.

2.2 Systèmes de Preuve par résolution

La technique de preuve par résolution est importante car elle est à la base des systèmes de programmation logique qui permet de calculer à partir de programmes décrits de manière logique par des clauses.

La règle de résolution ou principe de résolution de Robinson (1965) est l'application la plus fondamentale de la logique propositionnelle. Elle est principalement utilisée dans les systèmes de preuve automatiques. Cette technique de preuve par résolution est très importante car elle est à la base des systèmes de programmation logique qui permet de calculer à partir de programmes décrits de manière logique par des clauses.

Appliqué à une formule, il nous permet de dériver des clauses à partir des clauses de la formule.

Définition 2.1 (dérivation par résolution)**.** *Soient $\mathcal{F}$ une formule* CNF *et c une clause, on dit que c est une dérivation par résolution à partir de $\mathcal{F}$ s'il existe une séquence $\pi = [c_1, c_2, ..., c_n = c]$ telle que $\forall i \in (1, k]$, soit (1) $c_i \in \mathcal{F}$, soit (2) $c_k = \eta[x, c_i, c_j]$ avec $1 \leq i, j < k$.*

On appelle *réfutation* toute dérivation par résolution d'une clause vide $(c = \bot)$ à partir d'une formule $\mathcal{F}$, dans un tel cas, la formule $\mathcal{F}$ est insatisfiable.

Les trois règles présentées précédemment (voir 1.2.2, 1.2.2, 1.2.2) permettent d'établir une méthode de démonstration automatique complète pour la résolution. La règle de résolution 1.2.2 seule est complète pour la réfutation. En effet, si l'ensemble de clauses considéré est inconsistant on arrive toujours à générer la clause vide.

Généralement, la séquence de dérivation est réprésentée par un graphe de résolution qui est orienté et acyclique (DAG en anglais).

Définition 2.2 (graphe de résolution)**.** *Soient $\mathcal{F}$ une formule et c la clause que l'on souhaite dériver. Un graphe de résolution est un arbre tel que :*

- *chaque feuille est étiquetée par une clause de $\mathcal{F}$;*
- *chaque nœud qui n'est pas une feuille a deux fils et est étiquetée par une résolvante des clauses qui étiquettent ses fils ;*
- *la racine de l'arbre est la clause dérivée c.*

Exemple 2.1. *Soient $\mathcal{F} = \{(a \vee \neg d \vee \neg c), (c \vee \neg b), (b \vee d)\}$ et $\alpha = (a \vee c \vee \neg b)$. La Figure 2.1 représente sous forme arborescente la séquence de dérivation $[(a \vee \neg d \vee \neg c), (c \vee \neg b), (a \vee \neg d \vee \neg b), (b \vee d), (c \vee d), (a \vee c \vee \neg b)]$.*

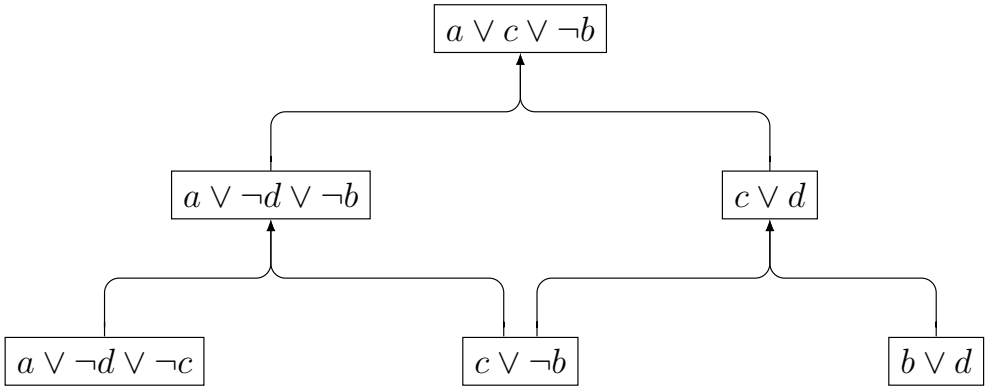


FIGURE 2.1 – Graphe de résolution.

Malgré sa simplicité, la résolution est difficile à mettre en œuvre efficacement, car il n'existe pas de méthode pour dire quelles résolutions effectuer et dans quel order pour arriver à la clause vide, ce qui peut entraîner facilement sur certains exemples la génération d'un nombre exponentiel de clauses sans jamais atteindre la clause vide, alors qu'on l'aurait obtenue en faisant les bons choix. Dans de tels cas, l'espace mémoire nécessaire devient très élévé et les temps de calculs prohibitifs ce qui est en pratique un obstacle majeur pour les solveurs SAT basés sur ce principe. Afin de permettre une amélioration pratique significative de la résolution, et éviter de générer des branches infinies, de nombreuses restrictions basées sur la structure de la preuve par résolution ont été proposées, elles sont généralement plus faciles à mettre en œuvre. Parmi celles-ci, nous pouvons citer :

la résolution régulière : Davis *et al.* (1962), Tseitin (1968), Galil (1977) le principe de cette approche repose sur la construction d'un arbre de preuve par résolution régulière. Un arbre de résolution est dit régulier s'il n'existe pas de chemin d'une feuille à la racine où une variable est éliminée plus d'une fois par résolution. En d'autres termes, sur chaque branche du graphe de résolution, l'ensemble des variables sur lesquelles on a effectué la résolution est sans répétition. L'instance est prouvée insatisfiable si et seulement s'il existe un arbre de preuve par résolution régulier ;

la P-résolution et la N-résolution : Robinson (1983) si une clause participant à la résolution contient uniquement des littéraux positifs (respectivement négatifs) alors l'étape de résolution est appelée P-résolution (respectivement N-résolution). Si la P-résolution est utilisée (respectivement N-résolution), seules les résolvantes faisant intervenir des clauses positives (respectivement négatives) sont ajoutées. Nous obtenons un gain dans le processus de résolution dans le sens où le nombre de clauses

strictement positives (respectivement négatives) est minime. Cependant, il est possible de trouver des formules qui se résolvent en temps polynomial en utilisant la résolution en général et qui utilisent un temps exponentiel en tenant compte de cette restriction. Malgré cela cette technique reste complète ;

la résolution étendue : Tseitin (1968) c'est une extension de la résolution à laquelle on ajoute la règle suivante : à chaque étape de la construction de la preuve, il est possible d'ajouter des lemmes à la formule, sous la forme d'une nouvelle variable y associée aux clauses qui codent $y \Leftrightarrow (l_1 \vee l_2)$ (*i.e.* $(y \vee \neg l_1), (y \vee \neg l_2)$ et $(\neg y \vee l_1 \vee l_2)$), où l_1 et l_2 sont deux littéraux apparaissant précédemment dans la preuve. Tout en étant d'apparence une règle très simple, l'ajout de nouvelles variables permet à la résolution étendue d'être très efficace Cook (1976) ; C'est un système de preuve plus puissant que la résolution standard car elle permet de résoudre en temps polynômial certaines classes d'instances que la résolution standard ne peut traiter qu'en temps exponentiel. Cependant, elle est très difficile à mettre en pratique car la règle d'extension accroit considérablement la taille de l'espace de recherche de la preuve.

En effet, la principale difficulté derrière l'automatisation du système de preuve par résolution étendue est de déterminer (1) quand est ce que de telles définitions seront ajoutées ? (2) quelles fonctions booléennes représenteront elles ? (3) comment les utiliser ?

Dés lors, cette automatisation reste encore une question ouverte.

Nous proposons au chapitre 4 une approche de résolution qui répond à la fois aux questions (1), (2) et (3).

la résolution linéaire : cette méthode a été indépendemment proposée par Loveland (1970), Luckham (1970) et Zamov et Sharonov (1969). Cette approche est un raffinement de la résolution classique permettant de réduire significativement le nombre de résolutions redondantes produites. Le principe consiste à restreindre la séquence de résolution afin de ne considérer que des dérivations linéaires. Une dérivation linéaire Δ, d'un ensemble de clauses $\mathcal{F}$, est une séquence de clauses $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ telles que $\alpha_1 \in \mathcal{F}$ et chaque α_{i+1} est une résolvante de α_i (le plus proche parent de α_{i+1}) et d'une clause β telle que (i) $\beta \in \mathcal{F}$, ou (ii) β est un ancêtre α_j de α_i où $j < i$;

la résolution unitaire : Dowling et Gallier (1984), Escalada (1989) la résolution unitaire est un cas particulier de la résolution où une des clauses à résoudre est une clause unitaire. Bien évidemment cette méthode n'est pas complète en générale, mais elle est complète pour les clauses de Horn ;

la résolution sémantique : Dennis (1994) la résolution sémantique est une extension de la résolution classique utilisant un modèle ou une interprétation afin de guider la recherche de preuve par réfutation. Plusieurs variantes de la résolution sémantique existent (pour un aperçu complet voir l'article de Dennis (1994)) ;

la résolution restreinte : cette approche consiste à ajouter à la formule un certain nombre de clauses produites par la résolution et à lancer un solveur basé sur la résolution régulière ensuite (ce dernier étant dans la plupart des cas beaucoup plus efficace que le principe de résolution). Diverses restrictions peuvent être mises en place. Ces restrictions sont plus ou moins complexes et plus ou moins bénéfiques pour DPLL (voir par exemple (Génisson et Siegel 1994, Castell 1996, Billionnet et Sutter 1992)) ;

Algorithme 2.1 : DP

Données : une formule CNF $\mathcal{F}$
Résultat : `vrai` si la formule est consistante, `faux` sinon

```
1 Début
2   tant que (V_F ≠ ∅) faire
3     x = ChoisirVariable(V_F);
4     V_F = V_F \ x;
5     F ⟵ F \ (F_x ∪ F_¬x);
6     F ⟵ F ∪ η[x, F_x, F_¬x];
7     si ∅ ∈ F alors retourner faux ;
8   retourner vrai ;
9 Fin
```

Une autre méthode décrite dans l'alogorithme 2.1 a été proposée par **Davis et Putnam** (1960). C'est l'une des premières méthodes de résolution dédiée au probème SAT. Elle utilise la propagation unitaire et l'élimination des littéraux purs. Son principe de base est l'application exhaustive de la résolution pour éliminer les variables. En effet, étant donnée une formule CNF $\mathcal{F}$ et une variable x de $\mathcal{F}$, soit $\mathcal{F}_x$(respectivement $\mathcal{F}_{\neg x}$) l'ensemble de clauses de $\mathcal{F}$ contenant le littéral x (respectivement $\neg x$). La procédure DP consiste à dériver toutes les résolvantes possibles entre $\mathcal{F}_x$ et $\mathcal{F}_{\neg x}$ notées $\eta[x, \mathcal{F}_x, \mathcal{F}_{\neg x}]$. Les deux sous-ensembles $\mathcal{F}_x$ et $\mathcal{F}_{\neg x}$ sont ensuite supprimées de $\mathcal{F}$ et remplacés par $\eta[x, \mathcal{F}_x, \mathcal{F}_{\neg x}]$. Ainsi, à chaque étape, le sous-problème généré contient une variable en moins, ce qui garantit la terminaison de l'algorithme. Mais la résolution peut augmenter le nombre de clauses de la formule. Par application successive de DP, un nombre exponentiel de clauses peut être ajouté dans le pire des cas. Cette méthode est complète et permet de prou-

ver la satisfiabilité ou l'insatisfiabilité d'un problème SAT sous forme CNF, suivant qu'elle ait produit durant ce processus une clause vide (le problème initial est insatisfiable) ou lorsque l'ensemble de clauses de la CNF est vide (le problème initial est satisfiable).

Dans la pratique, les méthodes basées sur la résolution sont très rarement utilisées dans leur forme originale (Chatalic et Simon (2000), Rish et Dechter (2000)). Cependant, une forme limitée de DP est à la base d'un des meilleurs pré-traitements de formule CNF, la méthode « SatElite » Eén et Biere (2005), intégré dans la plupart des solveurs SAT modernes. Cette forme limitée applique la résolution pour éliminer une variable uniquement si la taille de la formule n'augmente pas. En pratique, cette technique permet d'éliminer un nombre non négligeable de variables dans le cas des instances issues d'applications réelles.

2.3 Énumération

L'idée de base des algorithmes énumératifs est la construction d'un arbre binaire de recherche où chaque nœud représente une sous-formule obtenue à partir de la formule originale simplifiée par l'instanciation courante des variables.

Définition 2.3 (arbre binaire de recherche)**.** *Soit $\mathcal{F}$ une formule* CNF*, contenant l'ensemble des variables propositionnelles $\mathcal{V}_\mathcal{F} = x_1, x_2, ..., x_n$, l'arbre binaire de recherche correspondant à $\mathcal{F}$ est un arbre binaire où :*

- *chaque chemin de la racine à un nœud de l'arbre correspond à une interprétation partielle de $\mathcal{F}$.*
- *chaque arc est étiqueté par un littéral x ou $\neg x \in \mathcal{V}_\mathcal{F}$. La branche x_i (respectivement $\neg x_i$)Quine (1950) correspond à l'affectation de x_i à* `vrai` *(respectivement* `faux` *) ;*
- *les littéraux étiquetant les arcs issue d'un même nœud ont des valeurs opposées ;*
- *sur chaque branche de l'arbre, chaque variable apparaît une seule fois.*

Définition 2.4 (arbre complet)**.** *Soit $\mathcal{F}$ une formule* CNF *et $\mathcal{V}_\mathcal{F}$ l'ensemble des variables apparaissant dans $\mathcal{F}$. L'arbre binaire de recherche correspondant à $\mathcal{F}$ est dit complet si et seulement si, en plus des conditions exigées sur les arbres binaires de recherche, chaque chemin de la racine à une feuille de l'arbre correspond à une interprétation complète sur l'ensemble des variables de $\mathcal{F}$.*

Définition 2.5 (branche fermée)**.** *Une branche d'un arbre binaire de recherche correspondant à une formule* CNF *$\mathcal{F}$ est dite fermée si et seulement s'il existe un nœud N tel que l'interprétation partielle correspondante à celle-ci falsifie une des clauses*

de $\mathcal{F}$ et tel que les interprétations partielles de tout nœud ancêtre de N ne falsifie aucune clause de $\mathcal{F}$.

Définition 2.6 (arbre fermée)**.** *Un arbre binaire de recherche correspondant à une formule* CNF *$\mathcal{F}$ est dit fermé si et seulement si toutes ses branches sont fermées.*

Pour prouver la satisfiabilité d'une formule CNF $\mathcal{F}$, il suffit de trouver une feuille qui produit un modèle, c'est à dire une interprétation complète qui satisfait l'ensemble des clauses de la formule $\mathcal{F}$. Pour prouver l'insatisfiabilité d'un ensemble des clauses, il faut montrer que l'arbre correspondant est un arbre fermé. Comme l'arbre comporte 2^n feuilles, cette méthode n'est pas efficace du tout en pratique.

Exemple 2.2. *Soit $\mathcal{F} = \{(a \vee b \vee c), (\neg a \vee b), (\neg b \vee c), (\neg c \vee a), (\neg a \vee \neg b \vee \neg c)\}$ un ensemble de clauses, l'ensemble des variables est $\mathcal{V}_{\mathcal{F}} = \{a, b, c\}$ et l'arbre binaire de recherche correspondant à $\mathcal{F}$, où l'heuristique de branchement utilisée est l'ordre lexicographique, est illustré dans la figure 2.2.*

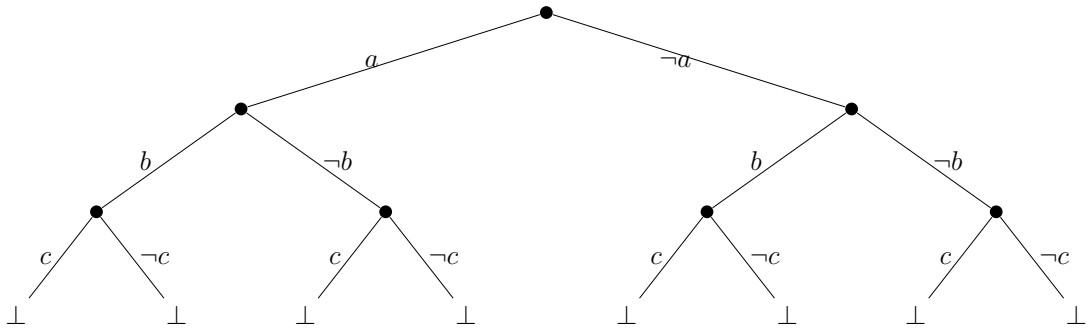


FIGURE 2.2 – Arbre binaire de recherche correspondant à l'ensemble de clauses de l'exemple 2.2.

2.4 Algorithme de Quine

L'algorithme de Quine (1950) consiste simplement à construire virtuellement l'arbre d'évaluation de la formule en procédant par des évaluations partielles et des simplifications. C'est une amélioration de la méthode des arbres sémantiques dans laquelle on réalise à chaque nœud de l'arbre binaire une évaluation partielle de la formule. Chaque nœud représente alors une sous-formule simplifiée par l'interprétation partielle courante. Si une évaluation partielle permet de conclure directement

à l'incohérence de la sous formule simplifiée, alors il n'est pas nécessaire de continuer la construction de l'arbre au delà de ce nœud. Cela nous permet de couper les branches alors dans certains cas, l'arbre sémantique construit selon l'algorithme de Quine peut être nettement inférieur à l'arbre sémantique complet. La méthode de Quine, décrit dans l'Algorithme 2.2, est basée sur la proposition suivante :

Proposition 2.1. *$\mathcal{F}$ est insatisfiable si et seulement si $\mathcal{F}_{|l}$ et $F_{|\neg l}$ sont insatisfiables.*

Algorithme 2.2 : QUINE

Données : $\mathcal{F}$ un ensemble de clauses
Résultat : `vrai` si la formule est consistante, `faux` sinon

```
1 Début
2   si (F = ∅) alors retourner vrai;
3   si (⊥ ∈ F) alors retourner faux;
4   l ⟵ HeuristiqueDeBranchement(F);
5   retourner (QUINE(F_|l) ou QUINE(F_|¬l))
6 Fin
```

Exemple 2.3. *Considérons la formule de l'exemple 2.2. L'arbre binaire construit implicitement par la méthode de Quine est illustré dans la Figure 2.3.*

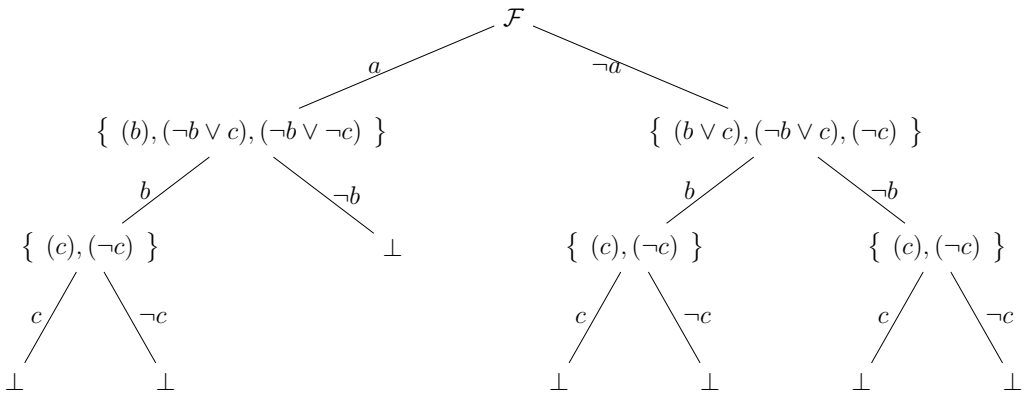


FIGURE 2.3 – Arbre construit par la méthode de Quine sur l'ensemble de clauses de l'exemple 2.2.

2.5 Procédure DPLL

En 1962, Martin Davis, George Logemann et Donald Loveland (Davis *et al.* (1962)) ont proposé une procédure appelée DPLL qui est une amélioration de la

procédure DP.

Avant de présenter l'algorithme DPLL, il est important de présenter tout d'abord la procédure de propagation unitaire, qui est l'un des procédés clés des algorithmes de résolution SAT et qui est sans conteste le plus utilisée.

2.5.1 Propagation Unitaire

La propagation unitaire(PU), aussi connue sous le nom de propagation de contraintes booléennes (**BCP**) est l'équivalent sémantique de la résolution unitaire. La PU représente la forme de simplification la plus utilisée et sans doute la plus utile des approches de type DPLL. Son principe de fonctionnement est le suivant : tant que la formule contient une clause unitaire, affecter son littéral à `vrai` . Cela repose sur la propriété élémentaire suivante :

Propriété 2.2. *Soit $\mathcal{F}$ une formule* CNF. *Si x est un littéral unitaire de $\mathcal{F}$, alors $\mathcal{F}$ est satisfiable si et seulement si $\mathcal{F}_{|x}$ est satisfiable.*

Ainsi $\mathcal{F}_{|x} = \{c | c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \setminus \{\neg x\} | c \in \mathcal{F}, \neg x \in c\}$, est la formule obtenue en éliminant les clauses contenant x et en supprimant $\neg x$ des clauses le contenant. Cette notion est étendue aux interprétations : soit une interprétation $p = \{x_1, ..., x_n\}$, on définit $\mathcal{F}_{|p} = (...((\mathcal{F}_{|x_1})_{|x_2})...{}_{|x_n}$. La propagation unitaire est l'application de cette simplification jusqu'à ce que la base des clauses ne contienne plus de clauses unitaires ou jusqu'à obtention d'une clause vide (contradiction). Formellement, nous avons :

Définition 2.7 (propagation unitaire)**.** *Soit $\mathcal{F}$ une formule* CNF*, nous notons $\mathcal{F}^*$ la formule obtenue à partir de $\mathcal{F}$ par application de la propagation unitaire. $\mathcal{F}^*$ est définie récursivement comme suit :*

1. *$\mathcal{F}^* = \mathcal{F}$ si $\mathcal{F}$ ne contient aucune clause unitaire :*
2. *$F^* = \bot$ si $\mathcal{F}$ contient deux clauses unitaires x et $\neg x$;*
3. *$F^* = (\mathcal{F}_{|x})^*$ tel que x est le littéral qui apparaît dans une clause unitaire de $\mathcal{F}$.*

Propriété 2.3. *Un ensemble de clauses $\mathcal{F}$ est satisfiable si et seulement si $\mathcal{F}^*$ est satisfiable.*

Définition 2.8 (déduction par propagation unitaire)**.** *Soit $\mathcal{F}$ une formule* CNF *et $x \in \mathcal{V}(\mathcal{F})$. On dit que x est déduit par propagation unitaire de $\mathcal{F}$, notée $\mathcal{F} \models_* x$, si et seulement si $(\mathcal{F} \wedge \neg x) \models \bot$. Une clause c est déductible par propagation unitaire de $\mathcal{F}$, si et seulement si $\mathcal{F} \wedge \bar{c} \models_* \bot$. i.e., $(\mathcal{F} \wedge \bar{c})^* \models \bot$.*

Propriété 2.4. *Soient $\mathcal{F}$ une formule* CNF *et $x \in \mathcal{L}(\mathcal{F})$ un littéral pur, x peut être propagé à* `vrai` *en préservant la satisfiabilité.*

Définition 2.9 (séquence de propagations)**.** *Soit $\mathcal{F}$ une formule* CNF*, $\mathcal{P} = \langle x_1, x_2, \ldots, x_n \rangle$ représente la séquence de propagations obtenue à partir de $\mathcal{F}$ telle que $\forall x_i \in \mathcal{P}$ la clause unitaire $(x_i) \in \mathcal{F}_{|\{x_1, x_2, \ldots, x_{i-1}\}}$*

Définition 2.10 (séquence de décisions-propagations)**.** *Soient $\mathcal{F}$ une formule et x un littéral de $\mathcal{F}$, $\mathcal{S} = \langle (x), x_1, x_2, \ldots, x_n \rangle$ représente la séquence de décisions-propagations obtenue par l'application de la propagation unitaire sur la formule $(\mathcal{F} \wedge x)$ telle que $\langle x_1, x_2, \ldots, x_n \rangle$ est une séquence de propagations obtenue à partir de $(\mathcal{F} \wedge x)$.*

À partir de la séquence de propagation et de la séquence de décisions-propagations, nous pouvons construire une pile de propagation $\mathcal{H} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$, où $\mathcal{S}_0$ est une séquence de propagations obtenue à partir de $\mathcal{F}$, $\mathcal{S}_1$ est une séquence de décisions-propagations obtenue à partir de $\mathcal{F}^*$ en considérant x_1, $\mathcal{S}_i$ $(1 < i \leq n)$ est une séquence de décisions-propagations obtenue à partir de $(\mathcal{F} \wedge \bigwedge_{j=1}^{i-1} x_j)^*$ en considérant le littéral x_i.

Exemple 2.4. *Soient $\mathcal{F} = \{(\neg a \vee b), (\neg b \vee c), (\neg d \vee \neg e), (e \vee f), (e \vee \neg f \vee \neg g), (\neg h \vee \neg i), (a), (l \vee \neg e)\}$ une formule et $\delta = [d, h]$ une séquence de décisions. La pile de propagations obtenue après l'affectation des littéraux de la séquence de décisions est $\mathcal{H} = \{\langle a, b, c \rangle, \langle (d), \neg e, f, \neg g \rangle, \langle (h), \neg i \rangle\}$.*

Ensuite nous pouvons introduire pour un littéral l de la pile de propagation les notions de niveau de propagation ($niv(l)$), de clause raison ($\overrightarrow{raison}(l)$) et d'explication ($exp(l)$).

Définition 2.11 (niveau de propagation)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{H} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ la pile de propagations obtenue à partir de $\mathcal{F}$ en appliquant la séquence de décisions $[x_1, x_2, \ldots, x_n]$ respectivement en niveau $1, 2, \ldots, n$ et l un littéral de $\mathcal{F}$. Si $\exists \mathcal{S}_i \in \mathcal{H}$ tel que $l \in \mathcal{S}_i$ alors $niv(l) = i$, $niv(l) = \infty$ sinon.*

Exemple 2.5. *Considérons la formule et la séquence de décisions de l'exemple 2.4. Nous avons $niv(a) = 0$, $niv(\neg i) = 2$ et $niv(l) = \infty$.*

Définition 2.12 (clause raison)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation à partir de $\mathcal{F}$ en appliquant la séquence de décisions $\delta = [x_1, x_2, \ldots, x_n]$ et l un littéral de $\mathcal{F}$. Si $l \in \delta$ alors $\overrightarrow{raison}(l) = \bot$, sinon $\overrightarrow{raison}(l) \in \mathcal{F}$ telle que $l \in \overrightarrow{raison}(l)$ et pour tous les autres littéraux $y \in \overrightarrow{raison}(l), \mathcal{I}(y) = \bot$ et y précède l dans l'interprétation $\mathcal{I}$.*

Remarque 2.1. *Il est possible d'associer plusieurs clauses raison à un littéral propagé. Néanmoins, il est usuel de n'en considérer qu'une seule.*

Définition 2.13 (explication)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation à partir de $\mathcal{F}$ en appliquant la séquence de décisions $\delta = [x_1, x_2, \ldots, x_n]$ et l un littéral de $\mathcal{F}$. Si $l \in \delta$ alors $exp(l) = \emptyset$, sinon $exp(l) = \{\neg x$ tel que $x \in \overrightarrow{raison}(l)$ avec $\overrightarrow{raison}(l)$ une clause raison de $l\}$.*

Remarque 2.2. *Comme pour les clauses raison il n'y a pas unicité de l'explication de la propagation d'un littéral.*

Exemple 2.6. *Considérons la formule et la séquence de décisions de l'exemple 2.4. Nous avons $\overrightarrow{raison}(g) = (e \vee \neg f \vee \neg g)$ et $exp(g) = \{\neg e, f\}$.*

2.5.2 Procédure DPLL

La procédure DPLL qui est décrite par l'algorithme 2.3 constitue le point de départ de nombreux travaux ayant pour objectif le développement de solveurs efficaces. C'est un algorithme énumératif dont le but est de générer un arbre de recherche. En effet elle structure dynamiquement un espace de recherche de toutes les affectations possibles dans un arbre bianire de recherche jusqu'à ce que, soit elle trouve une affectation satisfaisante, soit elle conclut qu'une telle affectation n'existe pas. Cette procédure est principalement basée sur le processus de propagation unitaire et peut être vue comme l'algorithme de Quine avec une étape supplémentaire d'inférence à chaque nœud de l'arbre : la propagation des litéraux unitaires (d'autres processus de filtrage sont possibles *hyperbin resolution* Bacchus (2002), simplification par littéraux purs, *etc.*, mais sont très rarement applicables de manière efficace en pratique). À chaque nœud, l'interprétation partielle courante est étendue par un littéral de décision et par une séquence de propagations. La variable est sélectionnée suivant une heuristique en utilisant des connaissances syntaxiques comme la longueur des clauses, le nombre d'occurrences des variables, etc. Par exemple, l'heuristique sélectionnant la variable apparaissant le plus souvent dans les clauses les plus courtes (MOMS : Most Occurrences in Minimal Size clauses) proposée par Freemann 1996. La différence entre les deux procédures DP et DPLL réside dans le fait que la première procède par élimination de variables en remplaçant le problème initial par un problème plus simple mais plus large, et la seconde procède par le principe, appelé *séparation*, qui consiste à choisir un littéral l de $\mathcal{F}$ et à décomposer la formule $\mathcal{F}$ en deux sous-formules $\mathcal{F} \wedge l$ et $\mathcal{F} \wedge \neg l$. DPLL est une méthode de type "choix+propagation".

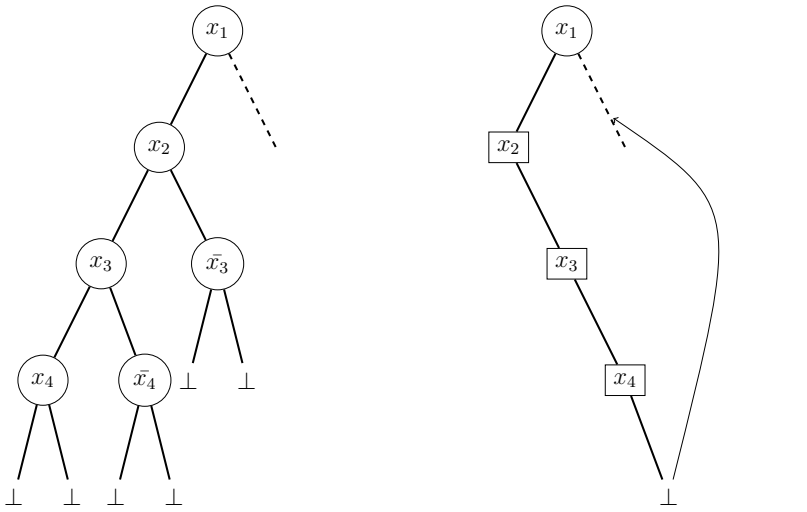
Exemple 2.7. *Soit la formule suivante :$\mathcal{F} = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$. La figure 2.4 illustre l'importance de la propagation unitaire dans*

Algorithme 2.3 : DPLL

Données : $\mathcal{F}$ un ensemble de clauses
Résultat : `vrai` si la formule est consistante, `faux` sinon

```
1 Début
2   F ⟵ SIMPLIFICATION(F);
3   si (F = ∅) ou (⊥ ∈ F) alors retourner (F = ∅) ou (⊥ ∉ F);
4   l ⟵ HeuristiqueDeBranchement(F);
5   retourner (DPLL(F ∧ l) ou DPLL(F ∧ ¬l))
6 Fin
```

une procédure DPLL. *En effet, dans le cas de* DPLL *(figure de droite) une décision* x_1 *suivie de la propagation unitaire est suffisante pour réfuter la décision d'affecter* x_1 *à* `vrai` *. Dans le cas de l'algorithme de Quine (figure de gauche), un sous-arbre a été développé avant de réfuter une telle décision.*



Quine(sans propagation) DPLL (avec propagation)

FIGURE 2.4 – La comparaison entre la méthode de Quine et de DPLL

L'exemple ci-dessous illustre le fonctionnement de l'algorithme DPLL sur l'instance de l'exemple 2.2. Les branches issues de littéraux unitaires et purs ne sont pas explorées.

Exemple 2.8. *Considérons la formule de l'exemple 2.2. L'arbre décrit par l'algorithme* DPLL *pour l'ensemble de clauses* $\mathcal{F}$ *est représenté dans la figure 2.5. Cet*

arbre n'admettant pas de feuilles représentant l'ensemble vide de clauses, $\mathcal{F}$ est démontrée inconsistante.

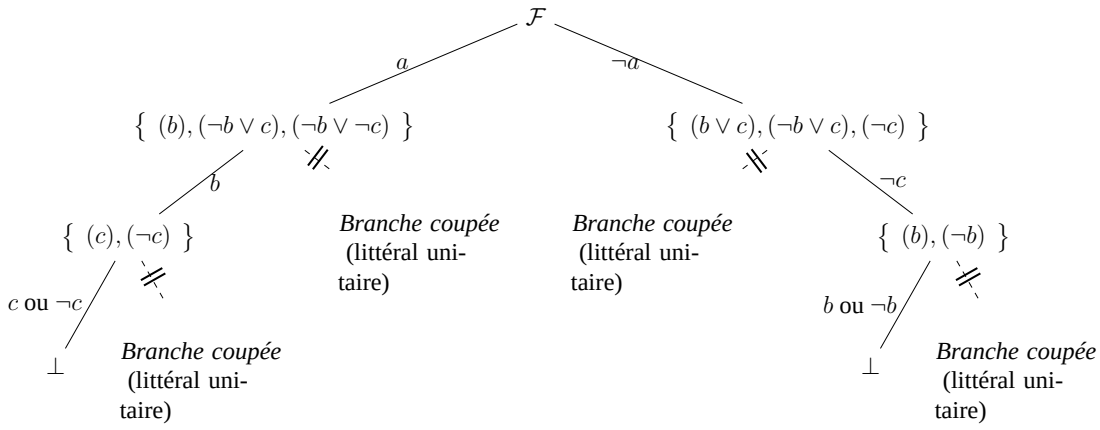


FIGURE 2.5 – Arbre de recherche construit par l'algorithme DPLL sur l'instance de l'exemple 2.2.

Par sa simplicité, la procédure DPLL est l'une des plus utilisées pour résoudre SAT et est actuellement la plus performante en pratique. Un point crucial quant à l'efficacité de la procédure DPLL est le choix de la variable de décision qui est effectué via une heuristique. Cette heuristique de branchement a une incidence directe sur la taille de l'arbre de recherche et sera présentée plus en détail par la suite (cf 2.5.3). De nombreuses améliorations ont été apportées à la procédure DPLL, elles concernent généralement un ou plusieurs des points suivants :

1. Heuristique de branchement : comment sélectionner la prochaine variable à affecter ?
2. simplification de la formule : comment simplifier la formule ?
3. traitement des échecs : que doit-on faire en cas de conflits ?

2.5.3 Heuristiques de branchements

L'ordre dans lequel les variables sont assignées par un algorithme de recherche est considéré depuis longtemps comme crucial. En effet, l'arbre de recherche exploré peut varier de manière exponentielle en fonction de l'ordre avec lequel les

variables sont affectées Li et Anbulagan (1997). De ce fait, une "bonne" heuristique de branchement est déterminante pour l'efficacité d'un algorithme de recherche. Cependant, sélectionner à chaque point de choix la variable qui permet de conduire à l'obtention d'un arbre de taille minimal est aussi un problème *NP-difficile* Liberatore (2000).

Au vu de la complexité théorique pour l'obtention de la variable de branchement optimale, il apparaît dès lors raisonnable d'estimer le plus précisément possible à l'aide d'une heuristique cette variable plutôt que de la calculer précisément. Cette heuristique, pour être efficace, doit permettre de diminuer la hauteur de l'arbre de recherche. De plus, un compromis entre le temps nécessaire au choix de la variable et le nombre de nœuds économisés doit être effectué. En effet, une heuristique trop gourmande en temps, même si elle réduit considérablement la taille de l'arbre de recherche, peut être moins performante qu'une heuristique beaucoup moins coûteuse en temps mais qui générera plus de nœuds.

Ces dernières années de nombreuses heuristiques ont été proposées dans la littérature. Ces heuristiques fonctionnent en deux étapes, la première étape c'est le choix de la variable à affecter. Lorsqu'une variable est choisie comme point de choix, la deuxième étape consiste à choisir sa valeur de vérité.

Heuristique de choix de variables

Comme nous l'avons déjà souligné précédemment, le choix de l'heuristique pour un algorithme de type DPLL est d'une importance capitale. En effet, un mauvais choix peut conduire à une exploration totale de l'arbre de recherche alors qu'un meilleur choix permettra de tronquer certaines branches, et dans le meilleur des cas nous ne parcourons qu'une unique branche. Nous distinguons généralement trois types d'heuristiques de choix de variables : 1) les approches syntaxiques qui permettent d'estimer le nombre de propagations résultant de l'affectation d'une variable ; 2) les approches prospectives (de type « look-ahead ») qui essayent de détecter et d'éviter de futures situations d'échecs ; 3) les approches rétrospectives (de type « look-back ») qui essayent d'apprendre à partir des situations d'échecs. Ces trois classes correspondent, pour les deux premières, aux améliorations apportées à l'étape de simplification de la formule et pour la dernière au traitement des échecs. Nous dressons ci-dessous une liste non exhaustive de quelques méthodes pour chacune de ces classes d'heuristiques.

Heuristiques « syntaxiques » Les heuristiques de branchements syntaxiques peuvent être vues comme des algorithmes gloutons dont le but est de sélectionner les variables qui, une fois affectées, génèrent le plus de propagations possibles ou per-

mettent de satisfaire le plus de clauses. Toutes ces heuristiques sont basées sur l'utilisation d'une fonction d'agrégation permettant d'estimer l'effet de l'affectation d'une variable libre. Parmi ces approches, citons :

BOHM : Cette heuristique, proposée par Buro et Kleine-Büning (1992), consiste à choisir la variable qui, pour l'ordre lexicographique, maximise le vecteur de poids $H_1(x), H_2(x), \ldots, H_n(x)$ avec $H_i(x)$ est calculé de la manière suivante :

$$H_i(x) = \alpha \times max(h_i(x), h_i(\neg x)) + \beta \times min(h_i(x), h_i(\neg x)) \tag{2.1}$$

où $h_i(x)$ est le nombre de clauses de taille i contenant le littéral x. Les valeurs de α et β sont choisies de manière heuristique. Dans (Buro et Kleine-Büning (1992)), les auteurs suggèrent de fixer $\alpha = 1$ et $\beta = 2$.

MOM : L'heuristique MOM « *Maximum Occurences in Clauses of Minimum Size* , » maximum d'occurrences dans les clauses de tailles minimales est une des heuristiques les plus simpleS. Proposée par Goldberg (1979), cette heuristique sélectionne la variable ayant le plus d'occurrences dans les clauses de plus petites tailles. Une variable x est choisie de manière à maximiser la fonction suivante :

$$(f^*(x) + f^*(\neg x)) \times 2^k + f^*(x) \times f^*(\neg x) \tag{2.2}$$

où $f^*(x)$ est le nombre d'occurrences du littéral x dans les clauses les plus courtes non satisfaites. La valeur de k, tout comme le fait de décider qu'une clause est courte, est donnée de manière heuristique. Le choix de la variable ici est motivé par le fait qu'il favorise la propagation unitaire.

Exemple 2.9. *Soit la formule* $\mathcal{F} = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_4 \vee \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\neg x_2 \vee x_5 \vee x_6) \wedge (x_2 \vee \neg x_5 \vee x_3)$

Les clauses de plus petite taille étant $(\neg x_1 \vee x_2)$ et $(x_1 \vee \neg x_5)$ alors l'heuristique MOM choisira la variable x_1.

Cette heuristique a été améliorée à plusieurs reprises, en essayant par exemple d'équilibrer les arbres produits par l'affectation d'une variable (Freeman (1995), Dubois *et al.* (1996), Dubois et Boufkhad (1996), Pretolani (1996)).

JW : L'heuristique de branchement proposée par Jeroslow et Wang (1990) est basée sur le même principe que l'heuristique MOM. Elle favorise la sélection des variables apparaissant dans les clauses de petites tailles. En effet, la possibilité qu'une variable soit sélectionnée par JW est inversement proportionnelle à la taille des clauses dans lesquelles elle apparaît. Les auteurs introduisent deux heuristiques, lesquelles sont analysées dans (Hooker et Vinay (1994), Barth (1995)), afin de fournir un poids aux variables. Le poids d'un littéral l de $\mathcal{F}$ est calculé à l'aide de la fonction suivante :

$$J(l) = \sum_{\alpha \in \mathcal{F} | l \in \alpha} 2^{-|\alpha|}. \tag{2.3}$$

La première heuristique (JW-OS) proposée consiste à sélectionner le littéral l qui maximise la fonction $J(l)$ et la seconde (JW-TS) consiste à identifier la variable x qui maximise $J(x) + J(\neg x)$, et à affecter la variable x à vrai, si $J(x) \geq J(\neg x)$, et de l'affecter à faux sinon.

Heuristiques de type « *look-ahead* » Les heuristiques MOM et JW sélectionnent une variable en fonction de la situation courante lors de la recherche. Il semble plus intéressant de sélectionner une variable en prévoyant à l'avance son influence sur l'évolution de la recherche. Ainsi les heuristiques de type « *look-ahead* » sont basées sur ce principe. En outre, ces heuristiques consistent à anticiper le résultat de l'affectation d'une variable non encore affectée à l'aide des méthodes de filtrages (telles que la propagation unitaire). En d'autres termes, elles prévoient l'évolution de la formule si une variable particulière est sélectionnée et effectuent une exploration de l'arbre de recherche en largeur, localement et temporairement. Dans le cas où une contradiction est détectée pendant la simplification d'une formule $\mathcal{F}$ par le littéral l, le littéral $\neg l$ est propagé au niveau courant. De cette manière, le nombre de variables éligibles pour le prochain choix de branchement est réduit. Ce type de méthodes fonctionnent surtout sur les problèmes là où l'apprentissage porte peu. Dans la suite, nous présentons une liste sommaire d'heuristiques de branchements basées sur ce concept :

PU : Les heuristiques basées sur la propagation unitaire (PU) Pretolani (1993), Freeman (1995), Li et Anbulagan (1997) utilisent la puissance de la PU afin de sélectionner plus précisément la prochaine variable à instancier. Cette méthode permet, contrairement à l'heuristique MOM, de considérer les propagations unitaires produites en cascade. Pour chaque variable libre x de la formule, $\mathcal{F}^*_{|x}$ et $\mathcal{F}^*_{|\neg x}$ sont calculés, la variable choisie est celle pour laquelle $|\mathcal{V}_{\mathcal{F}^*_{|x}}| + |\mathcal{V}_{\mathcal{F}^*_{|\neg x}}|$ est minimal. Cette méthode est plus discriminante, mais aussi plus coûteuse que MOM. Pour pallier ce problème, les méthodes qui l'exploitent réduisent son utilisation à un sous-ensemble

des variables non instanciées ou à la partie haute de l'arbre de recherche (Li et Anbulagan (1997)).

BSH : L'heuristique BSH (*Backbone Search Heuristic*) proposée par Dequen et Dubois (2004) et implémentée dans le solveur KCNFS est fondée sur la notion de *backbone*[3]. Plus précisément, l'heuristique proposée sélectionne les variables ayant le plus de chance d'appartenir au *backbone*. Pour cela, le score $score(x) = \text{BSH}(x) \times \text{BSH}(\neg x)$ de chaque variable x non assignée est calculé et la variable x qui maximise $score(x)$ est sélectionnée comme prochain point de choix. La valeur BSH d'un littéral l est calculée à l'aide de l'Algorithme 2.4 tels que : $\mathcal{B}(\updownarrow)$ représente l'ensemble $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ des clauses binaires tel que $\forall 1 \leq i \leq n$ le fait de falsifier α_i implique que l est satisfait, i représente le compteur permettant de contrôler le nombre de récursions possibles et $bin(l)$ (respectivement $ter(l)$) donne le nombre de clauses binaires (respectivement ternaires) possédant une occurrence de l.

Étant très gourmande en ressources, cette heuristique ne peut pas être utilisée avec un nombre important de récursions. Ce constat implique que le réglage du compteur i est très important. Afin de régler cette valeur de manière dynamique, les auteurs proposent de faire varier celle-ci en fonction de la hauteur du nœud considéré dans l'arbre de recherche. En effet, puisque les choix effectués en haut de l'arbre ont un impact très important sur la profondeur moyenne, il semble cohérent que la valeur de i en haut de l'arbre soit plus élevée qu'en bas.

Algorithme 2.4 : BSH($\mathcal{F}$: CNF, i : entier, l : littéral)

1 $i \longleftarrow i - 1$;
2 calculer $\mathcal{B}(\updownarrow)$;
3 **si** $i = 0$ **alors retourner**
$\sum_{(u \vee v) \in \mathcal{B}(\updownarrow)} (2 \times bin(\neg u) + ter(\neg u)) \times (2 \times bin(\neg v) + ter(\neg v))$;
retourner $\sum_{(u \vee v) \in \mathcal{B}(\updownarrow)} \mathsf{BSH}(\neg u) \times \mathsf{BSH}(\neg v)$;
4

Heuristiques de type « look-back » Les heuristiques de type « look-back » tirent parti des échecs pour revenir à des points de choix pertinents (*intelligent backtracking*) ou pour conserver une information clef provenant d'une longue phase de recherche et de déductions (enregistrement de *nogoods* ou pondération des variables).

3. Un littéral appartient au *backbone* d'une formule si et seulement s'il appartient à tous les modèles de la formule.

Une méthode basée sur ce concept, consiste à pondérer les variables apparaissant dans les derniers conflits. Cette approche permet, en particulier, de focaliser la recherche sur les parties difficiles du problème. Parmi les heuristiques s'appuyant sur ce principe, nous pouvons citer :

pondération des variables conflits : l'heuristique VSIDS (« *Variable State Independent Decading Sum* »), proposée par Zhang *et al.* (2001), est l'heuristique de branchement la plus utilisée dans les implantations modernes de DPLL. Cette heuristique associe un compteur à chaque variable appelé *activité*. Lorsqu'un conflit survient, une analyse du conflit (cf. 3.4) permet de déterminer la raison du conflit et les variables en cause dans ce conflit voient leur activité augmentée. Au prochain branchement, la variable ayant la plus grande activité est alors choisie comme variable de décision. En effet, puisque celle-ci est la plus impliquée dans les conflits, elle est donc jugée fortement contraignante. De façon à être précis, il est important de signaler que la pondération des variables est de plus et plus forte au fur et à mesure que le nombre de conflits augmente et l'activité des variables est divisée périodiquement par une certaine constante.

Le fait de pondérer plus fort les nouveaux conflits permet de privilégier les variables récemment intervenues dans les conflits et le fait de diminuer l'activité nous permet d'éviter le débordement de mémoire.

Plusieurs travaux récents sur le problème SAT sont basés sur l'heuristique VSIDS. Nous reviendrons plus en détails sur cette heuristique au chapitre suivant.

pondération des clauses conflits : cette approche proposée par Brisoux *et al.* (1999) est l'ancêtre de VSIDS. Elle est issue du constat suivant : lorsqu'une clause a été montrée insatisfiable dans une branche particulière de l'arbre de recherche, il est important que cette information ne soit pas négligée par la suite. Effectivement, il peut s'avérer intéressant de chercher à retrouver cette inconsistance le plus rapidement possible dans les autres branches de l'arbre de recherche. Une manière de réaliser ceci est de donner une priorité plus importante aux littéraux apparaissant dans la clause menant à l'échec. Pour cela, à chaque fois qu'une variable propositionnelle conduit à une inconsistance, la clause ayant amenée au conflit voit sont poids augmenter d'une certaine valeur. Un des avantages de cette méthode est de pouvoir être combinée facilement avec d'autres heuristiques de branchement telles que MOM ou JW. En effet, les poids induient sur les clauses peuvent être facilement transférés aux variables ;

Heuristiques de polarité

Une fois la variable choisie, il faut faire le choix de sa valeur de vérité (sa polarité). Ainsi, l'heuristique de choix de polarité qui est la deuxième étape de l'heuristique de branchement est aussi très importante pour la suite de la recherche. Elle n'est pas souvent évoquer dans la description des solveurs SAT. Quand une variable est choisie comme un point de choix, il faut lui associer une valeur de vérité, ce choix est important puisqu'il dirige la recherche. De part la complexité algorithmique, ce choix est effectué pour l'instant de mainière heuristique. Nous citons ici quelques unes des plus connues :

false/true : Cette heursitique très simple de phase consiste à toujours affecter la valeur de vérité fausse/vrai ($\neg x/x$) à la variable.

JW : Cette heuristique proposée dans (Jeroslow et Wang (1990)) permet de selectionner la polarité de la prochaine variable de branchement à partir d'une mesure w qui prend en compte des informations sur le problème (taille et nombre de clauses contenant la variable). Ici les auteurs proposent une fonction permettant de calculer le score de chaque variable. Cette fonction est identique à celle décrite lors de la présentation de l'heuristique du même nom (cf. équation 2.3) ;

progress saving : L'objectif de cette heuristique basée sur les polarités des littéraux, est d'empêcher le solveur de résoudre une sous-formule satisfiable plusieurs fois. Partant du constat que dans les solveurs CDCL, les redémarrages et backjumping peuvent conduire à la résolution répétitive des mêmes sous-formules, Pipatsrisawat et Darwiche (2007) proposent d'enregistrer dynamiquement pour chaque variable la dernière polarité utilisée pendant la recherche. Ces polarités mémorisées peuvent être représentées comme une interprétation complète notée $\mathcal{I}$. Ainsi, chaque fois qu'une variable de décision est choisie, sa polarité d'affectation est choisie dans $\mathcal{I}$ (choisir x si $x \in \mathcal{I}$, $\neg x$ sinon). De cette manière, l'effort pour satisfaire une sous-formule déjà résolue auparavant sera capitalisé. Bien que permettant d'intensifier la recherche, cette heuristique ne permet pas une bonne diversification de la recherche. Afin de remédier à ce problème, Biere (2009) propose de ne considérer qu'une partie de l'interprétation $\mathcal{I}$

occurence : Le principal objectif de l'heuristique *occurence* est d'équilibrer le nombre de littéraux positifs et littéraux négatifs pour permettre d'effectuer plus de résolutions. Elle a été proposée par Hamadi *et al.* (2009b) et a été implémentée dans le solveur LYSAT. Dans ce solveur, le poids d'un littéral l est mésuré par le

nombre d'occurrences $\#occ(l)$ de celle-ci dans les clauses apprises (choisir l si $\#occ(l) \geq \#occ(\neg l)$, $\neg l$ sinon).

2.5.4 Analyse de conflits et retour-arrière non chronologique

L'algorithme DPLL utilise une stratégie d'exploration avec un retour-arrière assez simple (chronologique). En effet, lors d'une detection de conflit, l'algorithme remonte dans l'historique des affectations à la variable de décision la plus récemment affectée. Le retour-arrière chronologique est relativement efficace pour des instances SAT aléatoires, en revanche il est assez insuffisant pour des instances indistrielles du fait de la non considération des raisons qui ont entrainer le conflit. Ce problème est pallié grâce à l'analyse de conflit qui permet d'identifier les affectations qui sont à l'origine des conflits, et de remonter dans l'historique jusqu'à une des causes du conflit (la variable en cause la plus récemment affectée). Ainsi, les sous-arbres menant au même conflit ne seront pas explorés inutilement. Ce processus permet d'effectuer un retour-arrière à un niveau $m < n$ sans nécessairement essayer toutes les possibilités entre le niveau du conflit n et le niveau m. Les algorithmes de type DPLL ont été largement étudiés et appliqués dans plusieurs domaines de l'IA, comme les problèmes de satisfaction de contraintes CSP (Dechter (1990), Ginsberg (1993), Schiex et Verfaillie (1993)) les systèmes de maintien de vérité ATMS (McAllester (1980), Stallman et Sussman (1977)), et en programmation logique (Bruynooghe (1980)), *etc.*. La principale différence entre ces algorithmes résident dans les techniques utilisées pour analyser les échecs (l'analyse de conflits) et pour éviter que la recherche ne conduise ultérieurement à une situation identique. L'analyse de conflit fut introduite pour la première fois dans les solveurs SAT par l'intermédiaire de Marques-Silva et Sakallah (1996a). Elle permet d'inférer une clause dite de conflit qui résume les racines d'un conflit, c'est-à-dire l'affectation de certaines variables qui conduisent inévitablement à la dérivation de la clause vide Bayardo Jr. et Schrag (1997). Cette clause permet d'identifier la dernière décision dans la branche courante à l'origine du conflit. Un retour-arrière non chronologique est ensuite effectué pour remettre la valeur de vérité de cette variable.

Exemple 2.10. *Soit $\mathcal{F}$ une formule* CNF *définie comme suite :*

$$\mathcal{F} = \begin{cases} (c_1) & (a \vee b) \\ (c_2) & (b \vee c) \\ (c_3) & (\neg a \vee \neg x \vee y) \\ (c_4) & (\neg a \vee x \vee z) \\ (c_5) & (\neg a \vee \neg x \vee z) \\ (c_6) & (\neg a \vee x \vee \neg z) \\ (c_7) & (\neg a \vee \neg y \vee \neg z) \end{cases}$$

Le fait d'un retour-arrière non chronologique est représenté dans la figure 2.6.

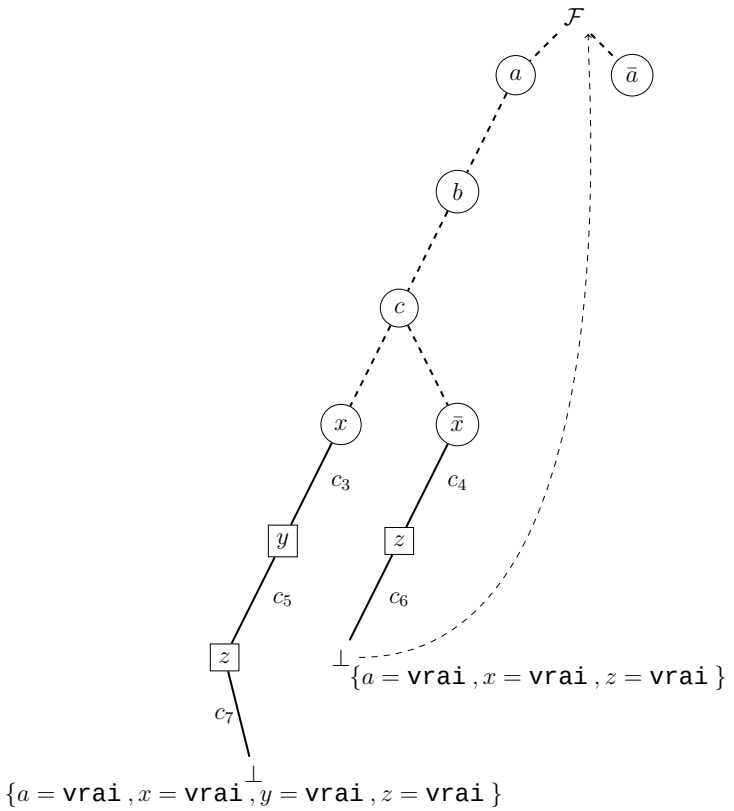


FIGURE 2.6 – Un retour-arrière non chronologique.

Les affectations de $a = \texttt{vrai}\,, b = \texttt{vrai}\,, c = \texttt{vrai}$ et $x = \texttt{vrai}\,,$ permettent de propager $y = \texttt{vrai}$ (clause c_3) et $z = \texttt{vrai}$ (clause c_5). Ces affectations et les propagations qui s'en suivent falsifient tous les littéraux de la clause c_7 qui devient vide. Dans cette situation, l'ensemble des affectations ayant provoqué l'échec à savoir $\{a = \texttt{vrai}\,, x = \texttt{vrai}\,, y = \texttt{vrai}\,, z = \texttt{vrai}\,\}$ constitue l'ensemble conflit. Il faut noter dans cet exemple que les littéraux b et c ne participent pas au conflit. L'algorithme essaie ensuite l'autre valeur de vérité de x. Supposons que l'algorithme détecte un nouveau conflit après avoir assigné x à la valeur de vérité `faux` , et que l'ensemble des affectations responsables est $\{a = \texttt{vrai}\,, x = \texttt{vrai}\,, z = \texttt{vrai}\,\}$. Une fois que toutes les valeurs de vérité de x ont été testées, le retour-arrière non chronologique peut alors se faire au niveau de a qui est assigné à `faux` et le processus recommence. Cette fois, on remarque que l'algorithme est en mesure de sortir du conflit sans essayer les décisions intermédiaires b et c.

L'idée derrière le retour-arrière non chronologique est qu'un retour-arrière sera effectué de manière à ce que l'état des assignations soit tel qu'il ne reste qu'un seul littéral non assigné. Son but principal est de permettre au solveur d'éviter partiellement le phénomène dit de « thrashing » (parcours répété des mêmes sous-arbres) Il peut toutesfois répéter les mêmes erreurs plus tard au cours de la recherche. Pour pallier à ce problème, on ajoute des clauses expliquant les raisons des conflits ("nogoods"). Cette clause, plus généralement connue sous le nom de clause apprise est obtenue en prenant la négation du terme construit par analyse de conflits $(\neg x_1 \vee \neg x_2 \ldots \neg x_n)$. Grâce à elle, il sera possible de continuer les différentes propagations alors que cela n'était plus possible sans elle. Ainsi à chaque fois q'un conflit est detecté, L'ajout de cette *clause apprise* (*learnt clause*) Marques-Silva et Sakallah (1996a), Bayardo Jr. et Schrag (1997), Zhang *et al.* (2001), Beame *et al.* (2004) permet d'indentifier et d'ajouter à la formule initiale une clause impliquée par la formule. De plus conserver cette clause pour la suite de la recherche permet de découvrir par propagation unitaire des nouvelles implications permettant s'esquiver les conflits déjà rencontrés.

2.6 Solveurs SAT modernes

Nous donnons dans cette section une brève description des solveurs SAT modernes. Une description plus formelle sera donnée dans le chapitre suivant (cf chapitre 3).

L'efficacité des solveurs SAT modernes est à l'origine des progrès impressionnants observés ces dernières années autour de la résolution pratique d'instances SAT industrielles. Ces solveurs SAT modernes ont été mis en œuvre grâce à une meilleure compréhension de la nature de la difficulté des problèmes (*i.e.* phénomène *heavy tailed* (Gomes *et al.* (1997; 2000)), *backdoor* (Williams *et al.* (2003b)), et *backbone* (Dequen et Dubois (2004))), à l'exploitation des propriétés structurelles des instances SAT (*i.e.* dépendances fonctionnelles (Grégoire *et al.* (2005))), à la mise en œuvre efficace des techniques d'apprentissage à partir des échecs (Marques-Silva et Sakallah (1996a), Bayardo Jr. et Schrag (1997), Zhang *et al.* (2001), Beame *et al.* (2004)), à l'introduction de nouveaux paradigmes algorithmiques (*i.e.* la randomization et les redémarrages (Gomes *et al.* (1998))), à la mise en œuvre d'heuristiques adaptatives dirigées par les conflits (Brisoux *et al.* (1999)) et à la proposition de structures de données paresseuses (*i.e. watched literals* (Zhang (1997), Zhang *et al.* (2001))). Malgré ces progrès certes impressionnants, ces solveurs ont toujours un comportement global qui se doit d'être relativisé.

La figure 2.7 illustre un schéma général d'un solveur SAT moderne.

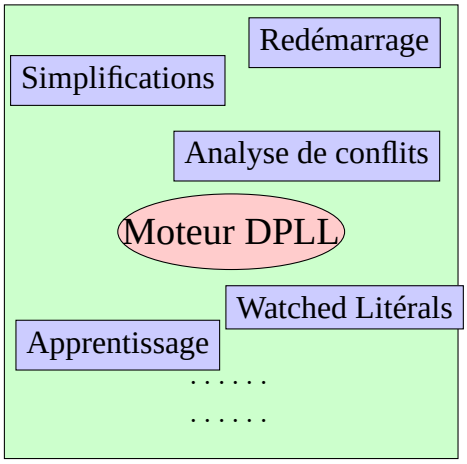


FIGURE 2.7 – Soleveur SAT moderne

Les solveurs SAT modernes (Marques-Silva et Sakallah (1996a), Zhang *et al.* (2001), Beame *et al.* (2004)) ne sont que le résultat d'un subtil mélange de ces principes au sein de l'algorithme DPLL. Cependant, plusieurs travaux effectués sur ces solveurs montrent que cette combinaison reste sensible aux changements mêmes legers.

2.7 Conclusion

Dans ce chapitre, nous avons présenté différents paradigmes pour résoudre de manière exacte le problème SAT. Comme nous l'avons montré ici, ces paradigmes s'appuient sur une recherche arborescente et utilisent des méthodes de propagation de contraintes afin de couper les branches ne pouvant conduire à une solution. Ces approches reposent pour la plupart sur la procédure DPLL et sur une analyse à partir des échecs rencontrés. Cette analyse de conflits et l'apprentissage de *no-goods* qui en découle permettent d'éviter de refaire plusieurs fois les mêmes erreurs dans l'arbre de recherche et d'apprendre une clause qui sera en contradiction avec l'état de l'arbre de recherche. Ces deux composants ont permi d'augmenter significativement la puissance des solveurs SAT tout en permettant aussi de résoudre des problèmes NP-complets. Le chapitre suivant est consacré à une description plus formelle des solveurs SAT modernes.

Chapitre 3

Solveurs SAT modernes

Sommaire

DANS CE CHAPITRE, nous présentons les solveurs de type CDCL en se focalisant sur leurs principales composantes. A savoir, le prétraitement, les structures de données paresseuses, l'apprentissage, l'heuristique adaptative, et pour terminer les redémarrages. Avant de conclure, Nous faisons une brève présentation des deux types d'approches utilisées pour la résolution du problème SAT en parallèle.

3.1 Architecture générale

Pour une bonne comprendre du fonctionnement des solveurs basés sur l'architecture CDCL, ansi que des interactions entre leurs différentes composantes, il est important que nous puissions tout d'abord présenter l'architectute générale de ces solveurs.

Algorithme 3.1 : Solveur CDCL

Données : $\mathcal{F}$ une formule sous CNF
Résultat : SAT or UNSAT

```
1  Δ ⟵ ∅;        /* base de donnée de clauses apprises */
2  I_p ⟵ ∅;                  /* interprétation partielle */
3  dl ⟵ 0;                        /* niveau de décision */
4  tant que (true) faire
5      α ⟵ PropagationUnitaire(F ∪ Δ, I_p);
6      si (α = null) alors
7          x ⟵ HeuristiqueDeBranchement(F ∪ Δ, I_p);
8          si (toutes les variables sont affectées) alors retourner SAT ;
9          l ⟵ChoisirPolarité(x);
10         dl ⟵ dl + 1;
11         I_p ⟵ I_p ∪ {(l^dl)};
12         si FaireReduction() alors ReductionClausesApprises
           (Δ);
13     sinon
14         si dl = 0 alors retourner UNSAT ;
15         β ⟵AnalyseConflit(F ∪ Δ, I_p, α);      /* une clause
           apprise */
16         bl ⟵AnalyseConflit(F ∪ Δ, I_p, α); /* niveau de saut
           arrière */
17         Δ ⟵ Δ ∪ β;
18         RetourArrière(F ∪ Δ, I_p, bl); /* mise à jour de I_p */
19         si (FaireRedémarrage()) alors Redémarrage(I_p, dl);
```

L'algorithme 3.1 définit le schéma général d'un solveur de type CDCL. Un tel solveur procède par décision+propagation. Un littéral de décision (lignes 7) est affecté suivant une certaine polarité (ligne 9) à un niveau de décision (dl). Si tous les littéraux sont affectés, alors $\mathcal{I}$ est un modèle de Σ (lignes 8). À chaque fois qu'un conflit est atteint par propagation unitaire (ligne 13-19), il est possible de prouver l'inconsistance de la formule (ligne 14) si le niveau de décision est 0. Si ce n'est pas le cas, un *nogood* β est calculé en utilisant une méthode d'analyse de conflits

donnée (ligne 15), le plus souvent selon le schéma « First UIP » (« *Unique Implication Point* » Zhang *et al.* (2001)), permettant de déduire qu'un littéral x aurait dû être propagé à un niveau plus haut dans l'arbre de recherche. Ce niveau correspond au niveau de *backjump* bl et est calculé en fonction du *nogood* β (ligne 16). Un saut arrière est alors effectué à ce niveau et le littéral x est propagé au niveau *backjump* (ligne 18). Au bout d'un certain nombre de conflits, les solveurs CDCL effectuent des redémarrages (voir Huang (2007) par exemple) qui consistent à recommencer une nouvelle recherche, en utilisant une nouvelle interprétation, depuis la racine de l'arbre de recherche (ligne 19). Afin d'être efficace, la base des clauses apprises peut être réduite selon une heuristique lorsque celle-ci est considérée comme trop volumineuse (ligne 12). Ces différentes composantes *Propagation Unitaire, Heuristique de Branchement, Apprentissage, Réduction de la base des clauses apprises, Heuristique de polarité, Heuristique de redémarrage etc.* sont nécessairement corrélées. Par exemple, l'heuristique de branchement et l'heuristique de polarité dirigent la propagation unitaire. La propagation unitaire va ensuite conduire l'apprentissage, et l'apprentissage en revanche peut influencer l'affectation des variables.
Afin de mieux comprendre les interactions entre les différentes composantes d'un solveur CDCL, nous présentons ces éléments dans un diagramme (voir figure 3.1)

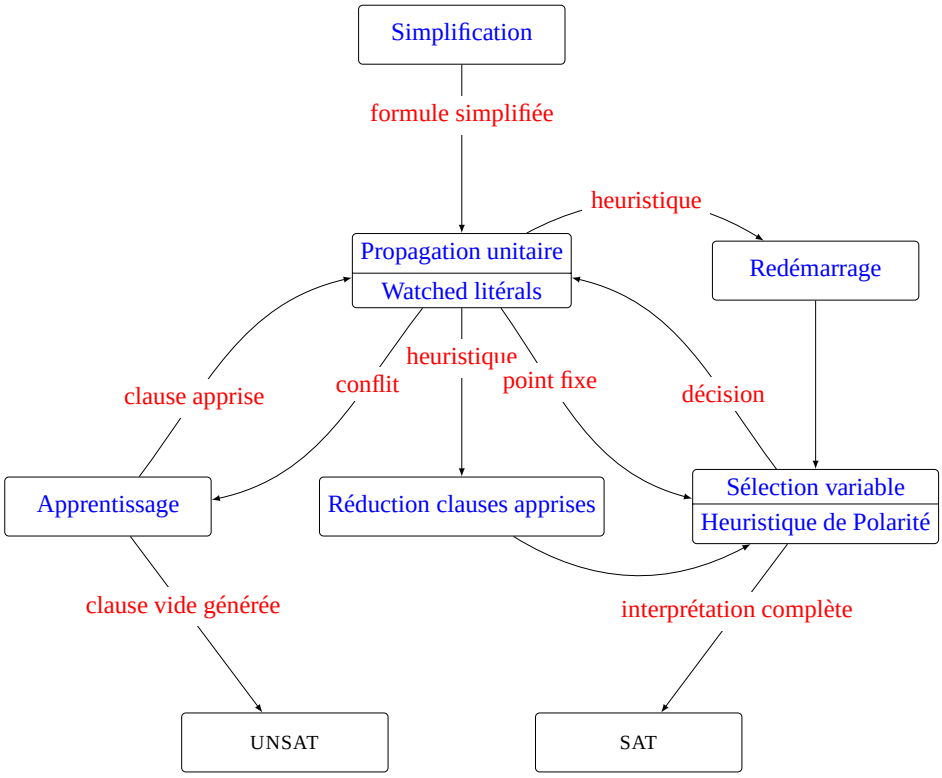


FIGURE 3.1 – Interaction des composantes d'un solveur CDCL.

Ce diagramme commence par une composante importante des solveurs SAT modernes qui est le prétraitement de la formule. Ce processus de prétraitement est présenté dans la section suivante.

3.2 Prétraitement de formules

Le problème SAT a été prouvé NP-complet Cook (1971), c'est-à-dire que pour une formule comportant n variables tous les algorithmes connus à ce jour ont une complexité en $\mathcal{O}(2^n)$ dans le pire de cas. Ce temps d'exécution dépend du nombre de variables mais pas du nombre de clauses. En théorie, réduire le nombre de variables implique une diminution de la complexité dans le pire de cas. Cependant, en pratique, le temps de calcul n'est pas entièrement corrélé au nombre de variables. Le nombre de clauses influence fortement les performances de la propagation unitaire ainsi que la fréquence d'apprentissage de clauses et les retours-arrières. Donc, éliminer des variables en ajoutant des clauses ne garantie pas une accélération de la résolution et reste problématique.

Les méthodes de prétraitement ont alors pour but de simplifier la formule par la réduction du nombre de variables si cette opération n'augmente pas la taille de la formule. Des approches de prétraitement peuvent être séparées en deux catégories. La première concerne des techniques qui conserve l'équivalence par rapport à la formule originale, la deuxième garantie uniquement l'équi-satisfiabilité. Des approches telles que la propagation unitaire, la subsumption ou la *self-subsumption* conservent l'équivalence Heule *et al.* (2010) et ne seront pas décrites dans cette partie. Elles sont appliquées dans la plupart des algorithmes de prétraitement Eén et Biere (2005) et permettent de supprimer des clauses et des littéraux redondants.

En plus de la propagation des littéraux purs et unitaires. De nombreuses techniques de prétraitement ont été introduites et utilisées. Parmi elles, nous citons ici :

Élimination de variables : L'élimination de variables, implémentée dans SATELITE Eén et Biere (2005), Subbarayan et Pradhan (2005), est une technique qui consiste à supprimer des variables de la formule initiale. Ce préprocesseur a été conçu pour comprimer assez vite une formule CNF pour ne pas être un goulot d'étranglement.

L'élimination d'une variable est obtenue par l'application de la résolution sur toutes les clauses où cette variable apparaît. Cette technique peut entraîner la création d'un nombre exponentiel de nouvelles clauses (exponentiel dans le nombre

de variables de la formule). Afin de limiter cette explosion combinatoire, les algorithmes de prétraitement n'appliquent cette élimination que si la suppression d'une variable n'augmente pas la taille de la formule origianle.

La formule obtenue par élimination de variables par résolution ne préserve pas l'équivalence logique mais uniquement l'équivalence pour la satisfiabilité. Eén et Biere (2005) ont montré que tout modèle de la nouvelle formule peut être étendue pour obtenir un modèle de la formule originale.

Élimination des clauses bloquées : Une clause $\alpha \in \mathcal{F}$ est dite bloquée si elle contient un littéral l bloqué. Un littéral $l \in \alpha$ est bloqué si $\forall \alpha' \in \mathcal{F}$ telle que $\neg l \in \alpha'$ la résolvante $\eta[l, \alpha, \alpha']$ est tautologique Heule *et al.* (2010), Järvisalo *et al.* (2010). Les clauses bloquées sont des clauses redondantes. Leur élimination dans une formule CNF permet de réduire taille de la formule initiale tout en conservant sa nature en terme de satisfiabilité. Cependant, cette élimination ne permet pas de conserver l'équivalence de la formule Heule *et al.* (2010). La suppression des clauses bloquées est effectuée jusqu'à l'obtention d'un point fixe. Notons que l'ordre de suppression n'a aucune importance puisque la méthode est confluente Järvisalo *et al.* (2010). De plus, puisque supprimer l'ensemble des clauses bloquées n'a pas de réelle influence sur les performances des solveurs, les méthodes de prétraitement n'utilisent pas les littéraux possédant de nombreuses occurrences ;

Élimination de clauses tautologiques dissimulées : L'élimination de clauses tautologiques dissimulées, proposée par Heule *et al.* (2010), est basée sur l'extension des clauses par ajout de littéraux, appelés littéraux dissimulés. L'ajout d'un littéral $l' \in \mathcal{L}_{\mathcal{F}}$ à une clause $\alpha \in \mathcal{F}$ est possible si $\exists l \in \alpha$ tel que $(l \vee \neg l') \in \mathcal{F} \setminus \{\alpha\}$. Cette extension est appliquée jusqu'à l'obtention d'un point fixe. Les auteurs montrent que si les clauses obtenues par l'ajout de tels littéraux sont tautologiques alors elles peuvent être supprimées de la formule initiale. Notons que l'application de cette technique permet d'obtenir une formule équivalente à la formule initiale Heule *et al.* (2010). L'ajout d'un littéral dissimulé est l'opération opposé de *self-subsumption* présenté au chapitre 1 ;

Élimination des équivalences : Selon la théorie de la complexité, le problème SAT étant exponentiel en fonction du nombre de variables de la formule, réduire leur nombre permet d'augmenter la vitesse de résolution. Pour effectuer cela, une méthode simple consiste à supprimer les littéraux équivalents. De tels littéraux peuvent être détectés simplement par la recherche de cycles dans le graphe d'implications obtenu à l'aide de clauses binaires de la formule. Une autre méthode permettant de

trouver les équivalences consiste à utiliser la technique de probing et à comparer l'affectation des littéraux.

Probing : Le *probing* est une technique, proposée par Lynce et Marques-Silva (2003), permettant la simplification d'une formule par l'application de la propagation unitaire sur les deux polarités d'une même variable. Les auteurs proposent trois approches permettant d'extraire des informations des interprétations partielles ainsi obtenues. La première consiste à vérifier que l'application de l'une des deux polarités ne conduit pas à une interprétation incohérente, ce qui permet le cas échéant de propager l'un des deux littéraux. Afin d'illustrer les deux autres méthodes. Considérons une formule $\mathcal{F}$, une variable l de $\mathcal{F}$, $\mathcal{I}_l = (\mathcal{F}_{|l})^*$ et $\mathcal{I}_{\neg l} = (\mathcal{F}_{|\neg l})^*$ les deux interprétations partielles obtenues par l'application de la propagation unitaire. La seconde approche consiste à propager les littéraux qui apparaissent dans les deux interprétations partielles. En effet, supposons que $x \in \mathcal{I}_l \cap \mathcal{I}_{\neg l}$ alors $\mathcal{F} \models (x \vee l)$ et $\mathcal{F} \models (x \vee \neg l)$ [4] et donc $\mathcal{F} \models x$. La dernière approche est celle qui nous intéresse puisqu'elle permet de détecter des équivalences. Pour cela, il suffit de vérifier s'il n'existe pas de littéraux apparaissant de manière complémentaire dans $\mathcal{I}_l$ et $\mathcal{I}_{\neg l}$. En effet, soit x tel que $x \in \mathcal{I}_l$ et $x \in \mathcal{I}_{\neg l}$ alors $(l \Rightarrow x)$ et $(\neg l \Rightarrow \neg x)$ et donc $(l \Leftrightarrow x)$;

Vivification : Vivification proposée par Piette *et al.* (2008) est une technique visant à raccourcir les clauses de la formule. Elle consiste à supprimer des littéraux de la clause en se basant sur la propagation unitaire.

La réduction d'une clause $\alpha = (l_1 \vee l_2 \vee \cdots \vee l_n)$ de $\mathcal{F}$ est obtenue en appliquant la propagation unitaire sur les littéraux complémentaires de α jusqu'à obtention de l'un des cas suivant :

1. $\mathcal{F}_{|\{\neg l_1, \neg l_2, \ldots, \neg l_i\}} \models_* \bot$ avec $i < n$. Dans ce cas la clause α peut être remplacée par la clause $\alpha' = (l_1 \vee l_2 \vee \cdots \vee l_i)$. En effet, puisque l'interprétation $\{\neg l_1, \neg l_2, \ldots, \neg l_i\}$ falsifie $\mathcal{F}$, la clause α' peut être ajoutée à la formule initiale. Comme α' subsume α, il est possible de remplacer α par α' ;
2. $\mathcal{F}_{|\{\neg l_1, \neg l_2, \ldots, \neg l_i\}} \models_* l_j$ tel que $l_j \in \alpha$ et $i < j < n$. Ici, le fait que $\mathcal{F}_{|\{\neg l_1, \neg l_2, \ldots, \neg l_i\}} \models_* l_j$ nous permet d'inférer la clause $\alpha' = (l_1 \vee l_2 \vee \cdots \vee l_i \vee l_j)$ laquelle subsume α. De la même manière que pour le premier cas, la clause α peut être remplacée par α' ;
3. $\mathcal{F}_{|\{\neg l_1, \neg l_2, \ldots, \neg l_i\}} \models_* \neg l_j$ tel que $l_j \in \alpha$ et $i < j \leq n$. Dans ce cas, $\mathcal{F}_{|\{\neg l_1, \neg l_2, \ldots, \neg l_i\}} \models_* \neg l_j$ nous permet d'inférer la clause $\alpha' = (l_1 \vee l_2 \vee \cdots \vee l_i \vee \neg l_j)$. Puisque, α' self-subsume α il est possible de remplacer la clause α par $\eta[l_j, \alpha, \alpha']$.

4. Si $\mathcal{F}_{|l} \models_* x$ alors $\mathcal{F}_{|l \wedge \neg x} \models_* \bot$ et donc $\mathcal{F} \models_* (\neg l \vee x)$

Ce rôle primordiale des techniques de simplifications a contribué inconstestablement au succès des solveurs SAT modernes. Ce qui a ouvert une autre voie de recherche visant à utiliser ces techniques pendant la recherche. En effet, pendant le processus de résolution la formule originale se retrouve transformée à cause des clauses apprises de façon générale et des clauses unitaires de manière particulière. Dès lors appliquer les différentes techniques de simplification citées auparavant après l'apprentissage de clauses unitaires permet d'améliorer la résolution du problème SAT.

Un autre problème posé par la résolution pratique du problème SAT est celui de la gestion mémoire de la formule pendant la recherche de solution. Pour résoudre ce problème, les solveurs SAT utilisent les structures de données paresseuses pour modéliser en mémoire la CNF permettant ainsi de mettre à jour beaucoup moins de clauses. Une description de ces structures est faite dans la section suivante.

3.3 Structures de données paresseuses ("Watched literals")

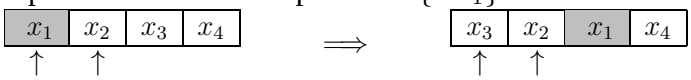
Comme nous l'avons souligné dans les sections précédentes, la propagation unitaire est un processus fondamental des solveurs SAT modernes. En efffet, l'expérience a montré que les solveurs SAT passent l'essentiel de leur temps à la propagation des clauses unitaires. Afin d'améliorer l'efficacité du processus de propagation untaire, de nombreux progrès algorithmiques ont été effectués ces dernières années, le principal est sans doute celui proposé par Zhang et Malik (2002).

Les auteurs introduisent une structure de données dite « paresseuse » qui permet un traitement de la propagation unitaire d'une complexité moyenne sous-linéaire. En effet, la constatation est d'autant plus évidente qu'un littéral ne peut être propagé tant qu'au moins deux littéraux de cette clauses ne sont pas affectés. Les auteurs ont proposé donc de garder deux pointeurs sur deux littéraux de chaque clause, appelés « *watched littérals* ». A chaque fois que l'un de ces littéraux est affecté à faux le pointeur est déplacé vers un autre littéral qui n'est pas faux. Lorsque les deux littéraux pointent le même littéral, cela implique que ce littéral est unitaire et pourra être propagé. L'intérêt de ces structures est double, elles permettent d'un coté de capturer la propagation unitaire efficacement et de l'autre de ne visiter que les clauses concernées par les affectations actuelles. Ces strutures permettent également d'éviter défauts de cache.

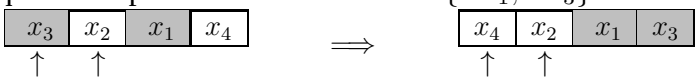
Dans *Picosat* Biere (2008b), l'auteur propose de fixer les deux pointeurs sur les deux premiers littéraux de chaque clause. A chaque fois qu'un de ces deux litté-

raux devient faux, au lieu de déplacer le pointeur, le solveur cherche dans le reste de la clause un autre littéral non affecté à faux et effectue une permutation. L'un des avantages majeurs de cette structure se situe en sa capacité à ne parcourir, lors d'une affectation d'un littéral x, que les clauses dans lesquelles $\neg x$ est pointé. Il est également important de noter qu'aucune mise à jour n'est nécessaire lors du retour-arrière.

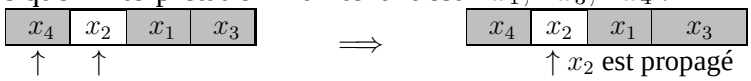
– commençons par considérer l'interprétation $\{\neg x_1\}$:



– supposons que l'interprétation maintenant est $\{\neg x_1, \neg x_3\}$:



– supposons que l'interprétation maintenant est $\neg x_1, \neg x_3, \neg x_4$:



Cette structure de données paresseuse posssède tout de même quelques inconvénients : Etant donné que les clauses ne sont pas considérées dans leurs intégralités, l'utilisation d'heuristiques syntaxiques n'est plus possible car ils requièrent une connaissance complète de l'instance après affectation(s). Nous perdons de l'information au profit de la vitesse. Seules les heuristiques sémantiques peuvent être appliquées avec les structures de données paresseuses, la structure du problème n'étant pas connue.

3.4 Apprentissage de clauses : analyse de conflit basée sur l'analyse du graphe d'implications

L'apprentissage des clauses est une des caractéristiques les plus importantes des solveurs SAT basés sur l'architecture CDCL. Elle permet d'apprendre de nouvelles clauses en cours de recherche, ensuite d'utiliser au maximum toutes les informations apprises dans l'espoir d'améliorer le temps de recherche. C'est la principale différence entre les procédures DPLL et CDCL. Les nouvelles clauses apprises étant généralement en contradictoire avec l'état de l'arbre de recherche, au cours de la recherche, chaque fois qu'une de ces clauses est apprise, les solveurs SAT réalisent un retour arrière non chronologique afin de s'assurer qu'au moins un de ces littéraux soit vrai. Le retour-arrière non chronologique permet au solveur d'éviter partiellement d'être soumis au « trashing » c'est à dire de redécouvrir les mêmes incohérences un grand nombre de fois. Mais il peut toutefois répéter les mêmes erreurs

dans le futur. La combinaison de ces techniques assure à la propagation unitaire d'être plus robuste à chaque apparition d'un nouveau conflit et permet au solveur de ne pas répéter les mêmes échecs. L'apprentissage des clauses à partir des conflits est effectué via un processus appelée *analyse de conflits*, qui est lançé dès l'apparition d'une clause falsifiée durant le processus de propagation unitaire. L'analyse de conflit vise à expliquer l'échec en analysant la trace de la propagation unitaire sur un graphe connu sous le nom de *graphe d'implications*.

3.4.1 Graphe d'implications

Le graphe d'implication est un graphe dirigé acyclique (DAG). C'est une structure de données centrale dans les solveurs CDCL. Il mémorise l'interprétation partielle en cours de construction ainsi que les implications faites. Pour effectuer cela, un nœud est associé à chaque littéral affecté à vrai par l'interprétation courante. Ensuite, une arête entre un nœud x et un nœud y est créée si l'affectation de x à vrai a impliqué l'affectation de y à vrai ($x \in exp(y)$). Formellemendt, le graphe d'implications se définit de la manière suivante :

Définition 3.1 (graphe d'implications)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation partielle. Le graphe d'implications associé à $\mathcal{F}$ et $\mathcal{I}$ est $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ tel que :*

- $\mathcal{N} = \{x \in \mathcal{I}\}$*, c'est-à-dire un nœud pour chaque littéral de $\mathcal{I}$, de décision ou propagé ;*
- $\mathcal{A} = \{(x, y)$ *tel que* $x \in \mathcal{I}, y \in \mathcal{I}$ *et* $x \in exp(y)\}$.

Exemple 3.1. *Soit la formule $\mathcal{F}$ suivante :*

$\alpha_1 = x_1 \vee x_2$ $\quad \alpha_4 = x_3 \vee x_5 \vee x_6$ $\quad \alpha_7 = \neg x_9 \vee x_{10} \vee x_{11}$ $\quad \alpha_{10} = x_{12} \vee x_{14}$
$\alpha_2 = \neg x_2 \vee \neg x_3$ $\quad \alpha_5 = \neg x_6 \vee x_7 \vee x_8$ $\quad \alpha_8 = x_8 \vee \neg x_{11} \vee \neg x_{12}$ $\quad \alpha_{11} = x_{12} \vee \neg x_6 \vee x_{15}$
$\alpha_3 = \neg x_2 \vee \neg x_4 \vee \neg x_5$ $\quad \alpha_6 = \neg x_4 \vee x_8 \vee x_9$ $\quad \alpha_9 = x_{12} \vee \neg x_{13}$ $\quad \alpha_{12} = x_{13} \vee \neg x_{14} \vee \neg x_{16}$
$\alpha_{13} = \neg x_{14} \vee x_{15} \vee x_{16}$

$\mathcal{I} = \{\langle(\neg x_1^1), x_2^1, \neg x_3^1\rangle, \langle(x_4^2), \neg x_5^2, x_6^2\rangle, \langle(\neg x_7^3), \neg x_8^3, \neg x_9^3\rangle, \langle(\neg x_{10}^4), x_{11}^4, \neg x_{12}^4, \neg x_{13}^4, x_{14}^4, x_{15}^4, x_{16}^4\rangle\}$ *est l'interprétation partielle obtenue par propagation de la séquence de décisions $\langle \neg x_1, x_4, \neg x_7, \neg x_{10}\rangle$. Le niveau de décision courant est 4. Nous représentons en rouge (respectivement vert) les littéraux de la formule falsifiés (respectivement satisfaits) par l'interprétation $\mathcal{I}$. L'interprétation $\mathcal{I}$ falsifie la formule (clause α_{13}). La figure 3.2 représente le graphe d'implications obtenu à partir de $\mathcal{F}$ et de l'interprétation $\mathcal{I}$.*

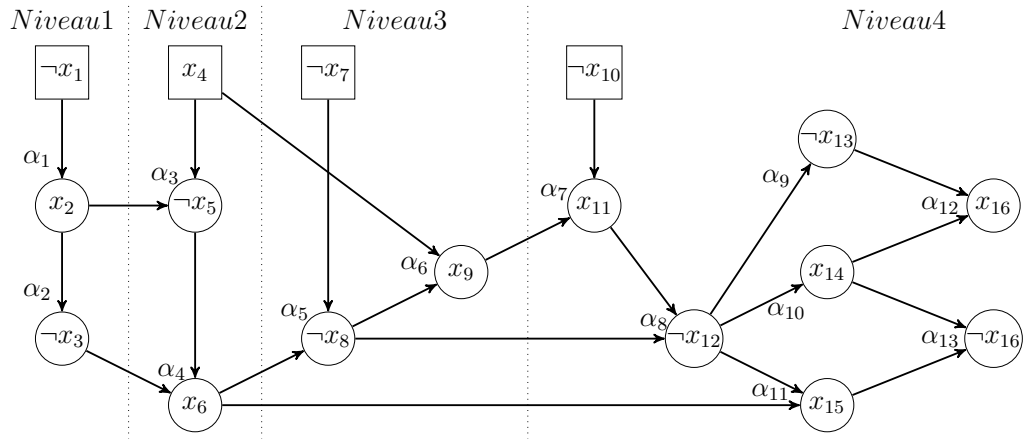


FIGURE 3.2 – Graphe d'implication obtenue à partir de la formule et l'interprétation partielle de l'exemple 3.1. Les nœuds de décision sont affichés en haut et sont représentés par des nœuds carrés. Les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'entre eux, la clause responsable de cette affectation est annotée. Le conflit se trouve sur la variable x_{16}.

3.4.2 Génération des clauses assertives

Dans cette section, nous définissons formellement le schéma d'apprentissage classique utilisé dans les solveurs SAT modernes. Dans les définitions suivantes, nous considerons $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une interprétation partielle telle que $(\mathcal{F}|_{\mathcal{I}})^* = \bot$ et $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ le graphe d'implications associé. Supposons que le niveau de décision courant est m, puisque l'on a atteint un conflit, le graphe d'implications $\mathcal{G}$ comporte deux nœuds représentant deux littéraux complémentaires, notons $z, \neg z$, on a $niv(z) = m$ ou $niv(\neg z) = m$. Alors il est possible d'analyser le graphe afin d'extraire les littéraux responsables de ce conflit. Généralement, cette analyse est basée sur le schéma UIP (*Unique Implication Point*). Un UIP est un nœud du niveau de décision courant qui domine les deux sommets correspondant aux littéraux conflictuels. Formellement, nous avons la définition suivante :

Définition 3.2 (nœud dominant un niveau)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle et $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ le graphe d'implications généré à partir de $\mathcal{F}$ et $\mathcal{I}$. Un nœud $x \in \mathcal{N}$ domine un nœud $y \in \mathcal{N}$ si et seulement si $niv(x) = niv(y)$ et $\forall z \in \mathcal{N}$, avec $niv(z) = niv(x)$, tous les chemins couvrent z et y passent par x.*

Exemple 3.2. *Considérons le graphe d'implications associé à l'exemple 3.1. Le nœud $\neg x_{12}$ domine le nœud x_{16} tandis que le nœud x_{14} ne le domine pas.*

Définition 3.3 (point d'implication unique (UIP))**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle conflictuelle et $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ le graphe d'implications généré*

à partir de $\mathcal{F}$ en fonction de $\mathcal{I}$. Le nœud x est un point d'implication unique si et seulement si x domine le conflit.

Remarque 3.1. *Les* UIP *peuvent être ordonnés en fonction de leur distance au conflit. Le premier point d'implication unique (*F-UIP *pour « First Unique Implication Point ») est l'*UIP *le plus proche du conflit tandis que le dernier* UIP *(*L-UIP *pour « Last Unique Implication Point ») est le plus éloigné, c'est-à-dire le littéral de décision affecté au niveau du conflit.*

Exemple 3.3. *Reprenons le graphe d'implications $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ généré dans l'exemple 3.1. Les nœuds $\neg x_{12}, x_{11}$ et $\neg x_{10}$ représentent respectivement le premier* UIP*, le second* UIP *et le dernier* UIP.

La localisation d'un UIP permet de fournir une décision alternative provoquant le même conflit. À l'heure actuelle, les prouveurs SAT modernes utilisent pour la plupart la notion de F-UIP afin d'extraire un *nogood* Zhang *et al.* (2001). Cette clause est générée en effectuant des résolutions entre les clauses responsables du conflit (utilisées durant la propagation unitaire) en remontant dans le graphe d'implications de la clause falsifiée vers la variable de décision du dernier niveau jusqu'à l'obtention d'une clause contenant un seul littéral du dernier niveau de décision (l'UIP). Ce processus est nommé preuve par résolution basée sur les conflits :

Définition 3.4 (preuve par résolution basée sur les conflits)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation partielle conflictuelle obtenue par propagation unitaire. Une preuve par résolution basée sur les conflits $\mathcal{R}$ est une séquence de clauses $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ qui satisfait les conditions suivantes :*

- *$\sigma_1 = \eta[z, \overrightarrow{raison}(z), \overrightarrow{raison}(\neg z)]$, tel que $\{z, \neg z\}$ est un conflit ;*
- *pour tout $i \in 2..k$, σ_i est construit en sélectionnant un littéral $y \in \sigma_{i-1}$ pour lequel $\overrightarrow{raison}(\bar{y})$ est défini. On a alors $y \in \sigma_{i-1}$ et $\bar{y} \in \overrightarrow{raison}(\bar{y})$, deux clauses pouvant entrer en résolution. La clause σ_i est définie comme $\sigma_i = \eta[y, \sigma_{i-1}, \overrightarrow{raison}(\bar{y})]$*
- *Enfin, σ_k est une clause assertive.*

Notons que chaque σ_i est une résolvante de la formule $\mathcal{F}$: par induction, σ_1 est la résolvante entre deux clauses qui appartiennent à $\mathcal{F}$; pour chaque $i > 1$, σ_i est une résolvante entre σ_{i-1} (qui, par hypothèse d'induction, est une résolvante) et une clause de $\mathcal{F}$. Chaque σ_i est aussi un *impliqué* de $\mathcal{F}$, c'est-à-dire : $\mathcal{F} \models \sigma_i$.

Ce processus permet d'obtenir une clause qui ne posséde qu'un seul littéral du niveau de décision courant. Cette clause que l'on nomme clause assertive est définie formellement comme suit :

Définition 3.5 (clause assertive)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation unitaire et m le niveau de décision courant. Une clause α de la forme $(\beta \vee x)$ est dite assertive si et seulement si $\mathcal{I}(\alpha) = \bot$, $niv(x) = m$ et $\forall y \in \beta, niv(y) < niv(x)$. Le littéral x est appelé littéral* assertif.

Il est important de souligner que dans le cadre des solveurs SAT modernes, les littéraux utilisés pour la résolution lors de la génération d'une preuve par résolution basée sur les conflits sont uniquement des littéraux du dernier niveau de décision. De plus, le littéral assertif obtenu à partir de la clause assertive générée par le biais de ce processus est un point d'implication unique (c'est ce processus qui est effectué par la fonction `analyseConflit`). Afin d'obtenir un F-UIP, il est nécessaire que la preuve par résolution soit élémentaire, ce qui est défini formellement comme suit :

Définition 3.6 (preuve élémentaire)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation unitaire et $\mathcal{R} = \langle \sigma_1, \sigma_2, \ldots \sigma_n \rangle$ une séquence de résolution basée sur les conflits. La séquence de résolution $\mathcal{R}$ est élémentaire si et seulement si $\nexists i < n$ tel que $\langle \sigma_1, \sigma_2, \ldots \sigma_i \rangle$ soit aussi une preuve par résolution basée sur les conflits.*

Exemple 3.4. *Considérons de nouveau la formule $\mathcal{F}$ et l'interprétation partielle $\mathcal{I}$ de l'exemple 3.1. La preuve par résolution basée sur les conflits est la suivante.*

$$
\begin{array}{lcll}
\sigma_1 & = & \eta[x_{16}, \alpha_{12}, \alpha_{13}] & = & x^4_{13} \vee \neg x^4_{14} \vee \neg x^4_{15} \\
\sigma_2 & = & \eta[x_{15}, \sigma_1, \alpha_{11}] & = & \neg x^2_6 \vee x^4_{12} \vee x^4_{13} \vee \neg x^4_{14} \\
\sigma_3 & = & \eta[x_{14}, \sigma_2, \alpha_9] & = & \neg x^2_6 \vee x^4_{12} \vee x^4_{13} \\
\sigma_4 & = & \eta[x_{13}, \sigma_3, \alpha_{10}] & = & \neg x^2_6 \vee x^4_{12} \\
\sigma_5 & = & \eta[x_{12}, \sigma_4, \alpha_8] & = & \neg x^2_6 \vee x^3_8 \vee \neg x^4_{11} \\
\sigma_6 & = & \eta[x_{11}, \sigma_5, \alpha_7] & = & \neg x^2_6 \vee x^3_8 \vee x^4_{10}
\end{array}
$$

Les littéraux assertifs des clauses assertives σ_4, σ_5 et σ_6 représentent respectivement le premier UIP*, le second* UIP *et le dernier* UIP.

La clause assertive correspondant au $1-$UIP ainsi générée est apprise par le solveur et permet non seulement d'éviter à l'avenir d'atteindre à nouveau cet échec, mais aussi d'effectuer un retour arrière. En effet, la connaissance de cette clause falsifiée permet d'affirmer que le littéral assertif doit être propagé plus tôt dans l'arbre de recherche. Ce calcul effectué à l'aide de la fonction `calculRetourArrière` est basé sur la propriété suivante :

Propriété 3.1 (clause assertive et retour arrière)**.** *Soient $\alpha = (\beta \vee x)$ une clause assertive déduite à partir de l'analyse du conflit tel que $i = max\{niv(\tilde{y})$ tel que*

$y \in \beta\}$. *Il est correct d'effectuer un retour arrière au niveau* i *et de propager le littéral* x.

Exemple 3.5. *Considérons la clause assertive* $\sigma = (\neg x_6^2 \vee x_{12}^4)$ *générée à partir de la formule* $\mathcal{F}$ *et l'interprétation partielle* $\mathcal{I}$ *de l'exemple 3.1. Le niveau de backtrack calculé à partir de cette clause est égal à* 2 *et l'interprétation partielle obtenue après avoir effectué un retour arrière au niveau* 2 *et avoir propagé le littéral assertif est* $\mathcal{I} = \{\langle(\neg x_1^1), x_2^1, \neg x_3^1\rangle, \langle(x_4^2), \neg x_5^2, x_6^2, x_{12}^2\rangle\}$.

Dans la littérature, plusieurs travaux supplémentaires portant sur le schéma d'analyse de conflit ont été proposés. Parmi ces travaux, nous pouvons citer ceux de Audemard *et al.* (2008a) où les auteurs proposent une extension du graphe d'implication. Cette extension est obtenue en considérant les clauses qui proviennent de la partie satisfaite de la formule. Ces clauses sont habituellement ignorées par l'analyse du conflit. Une autre approche qui a été proposée par Nadel et Ryvchin (2010) cherche à améliorer la hauteur du saut en effectuant un retour arrière à un niveau de décision inférieur à celui proposé par la clause assertive et à affecter et propager l'ensemble des littéraux de celle-ci. Bien d'autres approches Sörensson et Biere (2009), Han et Somenzi (2009), Hamadi *et al.* (2009a) permettant d'autres formes d'extensions ont été proposées.

Comme nous l'avons souligné plus haut, la clause assertive générée à partir de l'analyse de conflit permet de capturer une implication manquée à un niveau plus haut de l'arbre de recherche. Cependant il a été demontré dans la litérature qu'il existe d'autres implications manquées qui ne peuvent pas être decouvertes par les clauses assertives. En effet, pour générer la clause assertive, au cours du processus d'analyse de conflit, l'on passe par plusieurs clauses intermédiaires falsifiées parmi lesquelles celles contenant uniquement deux littéraux du niveau de décision courant. Dans Pipatsrisawat et Darwiche (2008b) les auteurs proposent d'apprendre au cours de l'analyse de conflit une nouvelle classe des clauses falsifiées contenant uniquement deux littéraux du niveau de décision courant. Ces clauses appelées clauses bi-assertives classiques qui sont plus courtes permettent de capturer les implications manquées par les clauses assertives. Elles sont dérivées en suivant le même processus de dérivation que celle utilisée pour générer la clause assertive. Cependant, il existe d'autres implications manquées qui ne peuvent pas être exprimées pas les bi-assertives dérivées du graphe d'implications en suivant la méthode classique de dérivation de la clause assertive. Dès lors, la question de trouver d'autres clauses permettant de capturer les implications manquées au cours du processus de propagation untaire à partir d'un graphe d'implications reste encore ouverte.

Nous proposons au chapitre 6 une nouvelle méthode d'analyse de conflit permettant de dériver une nouvelle classe de clauses bi-assertives qui tendent à être plus courtes et capturent plus d'implications que les clausses bi-assertives classiques.

Les clauses générées à partir du schéma d'analyse de conflits sont stockées dans une base appelée base de clauses apprises. Cependant le nombre des clauses générées peut être exponentiel. Dès lors la mémoire et la propagation unitaire se retrouvent affectées par l'explosion du nombre de clauses. Pour remédier à cela, la base de clauses apprises dans les solveurs SAT modernes est régulièrement réduite. La stratégie de réduction de cette base est généralement décrite comme suit : à chaque clause apprise est associé un score (selon un critère qui identifie son importance). Á chaque réduction, les clauses sont triées selon leur score et les clauses considérées peu importantes sont supprimées de la base des clauses apprises. Néanmoins, cette stratégie est heuristique et peut amener à supprimer des clauses essentielles pour la suite, ce qui peut s'avérer dramatique. En effet, conserver un grand nombre de clauses peut dégrader l'efficacité de la propagation unitaire, mais en supprimer trop peut affaiblir la puissance de l'apprentissage. Par conséquent, identifier les bonnes clauses apprises (c'est-à-dire importantes pour la preuve), définir une bonne fréquence de nettoyage et le nombre de clauses apprises à supprimer sont de véritables challenges.

Plusieurs travaux portant sur les critères d'évaluation de la qualité d'une clause apprise ont été proposés. Dans la section suivante, nous présentons quelques uns des ces critères.

3.5 Heuristiques de réduction de la base de clauses apprises

L'analyse de conflits et l'apprentissage ont apporté des améliorations significatives sur l'efficacité de l'algorithme DPLL Marques-Silva et Sakallah (1996a). Cependant, comme le font justement remarquer plusieurs travaux, il est nécessaire de gérer l'accroissement de la base de clauses apprises au risque de ralentir considérablement le processus de propagation unitaire Eén et Sörenson (2004). Pour éviter cela, tous les solveurs SAT modernes réduisent la base de clauses apprises (`reductionClausesApprises`) à l'aide d'une fonction heuristique.

Algorithme 3.2 : Stratégie de réduction

Données : Δ L'ensemble de clauses apprises de taille n
Résultat : Δ' L'ensemble de clauses apprises de taille $n/2$

```
1  si faireReduction() alors
2      TrierLesClausesApprises;     /* Selon les critères
       définis */
3      limit = n/2;
4      ind = 0;
5      tant que ind < limit faire
6          clause = Δ[ind] ;
7          si clause.size() > 2 alors
8              EliminerLaCLause;
9          sinon
10             GarderLaClause;
11 retourner Δ ;
```

L'algorithme 3.2 présente le schéma d'une stratégie de réduction.

Généralement, les stratégies de réduction de la base de clauses apprises procèdent en trois parties :

3.5.1 Clauses à conserver/enlever (lesquelles ?)

la première et aussi la partie la plus étudiée, consiste à trouver des critères pour juger de la qualité des clauses. La plupart des approches conservent systématique-

ment les clauses de taille deux et les clauses considérées comme raison d'un littéral propagé au moment de l'appel à la fonction réduction. Pour le cas des clauses binaires, le faite de les conserver par défaut peut être expliqué par le faite que les solveurs SAT modernes ont un comportement polynomial sur les formules constituées uniquement de clauses binaires. En ce qui concerne les clauses raisons ce n'est pas par soucis d'efficacité mais plus par nécessité qu'elles sont conservées. En effet, supprimer une clause apprise intervenant dans le processus de propagation unitaire ne permet plus d'assurer la construction du graphe d'implications et par conséquent l'analyse de conflits.

Dans la littérature, divers critères ont été proposés pour définir quelles sont les clauses susceptibles d'être pertinentes pour la suite de la recherche. Parmi ces critères, trois ont montré des résultats pratiques intéressants. La première, la plus populaire, est basée sur la notion de *first fail*. Ce critère, basée sur l'heuristique VSIDS, considère qu'une clause souvent impliquée dans les conflits est jugée plus pertinente. Le second est basé sur la mesure proposée dans Audemard et Simon (2009b) et appelée LBD (*Litetal Block Distance*). Le troisième critère appelé PSM a été proposé dans Audemard *et al.* (2011). Nous donnons plus de détails sur ces critères dans la suite.

VSIDS : Dans le solveur MINISAT Eén et Sörenson (2004) une activité est associée à chaque clause apprise. Cette activité est calculée de la même manière que l'heuristique de choix de variable VSIDS qui sera présentée dans la section suivante. Elle consiste à augmenter le poids des clauses impliquées dans le processus d'analyse de conflits. Afin de privilégier les clauses récemment touchées, comme pour l'heuristique de choix de variable VSIDS, il est nécessaire de tenir compte de leur âge. Pour effectuer cela, le solveur gère deux variables : inc initialisée à 1 et $decay < 1$. Lorsqu'une clause est utilisée dans le processus d'analyse de conflits, l'activité de celle-ci est augmentée de la valeur inc. Ensuite, après avoir mis à jour l'activité de l'ensemble des clauses ayant participées au conflit, la variable inc est augmentée telle que $inc = inc \div decay$.

LBD : Le second critère est basé sur une mesure identifiée dans Audemard *et al.* (2008b) et appelée LBD dans Audemard et Simon (2009b). Initialement, cette mesure correspond au nombre de niveaux différents intervenant dans la génération de la clause apprise. Formellement la valeur LBD d'une clause se calcule comme suit :

Définition 3.7 (distance LBD (« *Literal Block Distance* »))**.** *Soient $\mathcal{F}$ une formule, α une clause et $\mathcal{I}$ une interprétation partielle associant un niveau d'affectation à chaque littéral de α. Alors la valeur de* LBD *de α est égal au nombre de niveaux de décision différents des littéraux de la clause α.*

Dans Audemard et Simon (2009b), les auteurs montrent expérimentalement que les clauses ayant une faible valeur de LBD sont importantes pour la suite de la recherche. Partant de ce constat, ils proposent d'utiliser cette mesure afin de donner un poids aux clauses apprises. Pour cela, lorsqu'une clause est générée par analyse de conflits son score est initialisé avec la valeur de LBD courante, c'est-à-dire calculée par rapport à l'interprétation courante. Ensuite, à chaque fois qu'elle est utilisée dans le processus de propagation unitaire son poids est ajusté dans le cas où la nouvelle valeur de LBD calculée vis-à-vis de l'interprétation courante est inférieure à la valeur actuelle.

PSM : Le troisième critère est basé sur une mesure proposée par Audemard *et al.* (2011). Cette mesure est basée sur la notion de *progress saving* Pipatsrisawat et Darwiche (2007), elle est dynamique et correspond à la distance de hamming entre l'interprétation courante et les polarités des variables de la clause apprise. Formellement la valeur de PSM d'une clause se calcule comme suit :

Définition 3.8 (mesure PSM)**.** *Étant donnée une clause α et une interprétation complète $\mathcal{I}$ représentant l'ensemble des polarités associé à chaque variable, nous définissons* $\text{PSM}_{\mathcal{I}}(\alpha) = |\mathcal{I} \cap \alpha|$.

Dans Audemard *et al.* (2011), les auteurs montrent expérimentalement que les clauses ayant une faible valeur de PSM sont importantes pour la suite de la recherche. De ce constat, ils proposent d'utiliser ce critère afin d'estimer l'importance de la clause apprise. Les valeurs de PSM pour chaque clause apprise sont calculées et ajustées à une certaine fréquence.

La plupart des solveurs SAT de l'état de l'art gardent systématiquement uniquement les clauses binaires, tandis que pour les clauses de taille supérieure à deux, plusieurs mesures sophistiquées de pertinence ont été proposées pour prédire la qualité des clauses plus pertinentes. Ainsi, différentes implémentations de solveurs SAT CDCL sont proposées chaque année à la compétition SAT, chacune utilisant les mesures de pertinence décrites ci-dessus pour prédire la qualité des clauses susceptibles d'être pertinentes pour la suite de la recherche. Ces implémentations sophistiquées augmentent la difficulté à comprendre ce qui est essentiel de ce qui n'est pas.

Dans le chapitre 7, nous révisitons les différentes stratégies proposées dans la littérature pour mesurer la pertinence de clauses. Ensuite, nous proposons plusieurs stratégies de suppression statiques et dynamiques basées sur la pertinence, qui nous permettent de garder les clauses apprises qui sont plus susceptibles de couper des branches au sommet de l'arbre de recherche.

3.5.2 Fréquence de nettoyage (quand ?)

Il découle de la section précédente que la base des clauses apprises doit être constamment néttoyée au cours de la recherche. Il est donc important de savoir la fréquence avec laquelle cette base doit être nettoyée.

Cette fréquence étant calculée de manière heuristique (fonction `faireReduction`) doit être bien gérée. En effet une mauvaise gestion de cette dernière pourra conduire à rendre le solveur incomplet. A l'opposé de la méthode DPLL, l'approche CDCL a besoin des clauses apprises afin de se souvenir de l'espace de recherche qu'elle a déjà exploré. Ceci implique que la suppression d'une clause apprise peut amener le solveur à parcourir plusieurs fois le même espace de recherche. Afin de pallier ce problème et de garantir la complétude de la méthode, il est nécessaire que l'intervalle de temps atteigne une taille telle que l'ensemble des clauses apprises pouvant être générées dans cet intervalle permet d'assurer la terminaison de l'algorithme quelque soient les clauses supprimées précédemment. Actuellement dans la littérature, deux approches sont principalement utilisées pour gérer cet intervalle. La première approche dont la stratégie consiste à appeler la fonction de réduction de la base de clauses apprises une fois que la taille de celle-ci atteint un certain seuil a été implémentée dans MINISAT Eén et Sörenson (2004). La deuxième approche implantée dans le solveur GLUCOSE Audemard et Simon (2009a) consiste à réduire la base de clauses apprises lorsque le nombre de conflits obtenu depuis le dernier nettoyage est supérieur à $4000 + 300 \times x$ où x représente le nombre d'appels à la fonction de réduction.

3.5.3 Nombre de clauses à conserver/enlever (combien ?)

Un autre problème souligné par la gestion de la base des clauses apprises est celui de savoir le nombre de clauses à enlever. En effet, dans la plupart de solveurs SAT modernes, lorsque la fonction de réduction est appelée, l'ensemble des clauses apprises est trié suivant différents critères et un un certain seuil est calculé à partir des critères utilisés (par exemple dans MINISAT, un seuil est calculé par le compteur d'augmentation et la taille de la base est définie). Ensuite, la moitié des clauses estimées comme moins importantes plus les clauses qui ont des valeurs en dessous du seuil sont supprimées.

3.6 Heuristiques de choix de variables de branchement (VSIDS)

L'évolution des heuristiques classiques a été marquée par l'apparition d'heuristiques dynamiques dirigées par les conflits et dont le coût de calcul est négligeable et qui présente la particularité d'être indépendantes de l'interprétation courante.

Dans Brisoux *et al.* (1999), les auteurs proposent d'associer une activité β_c à chaque clause c. A chaque fois qu'un conflit est détecté (un littéral et son complémentaire ont été capturé par la propagation unitaire). Les deux clauses impliquées dans le conflit voient leurs activités incrémentées. Ensuite l'activité du littéral à choisir est calculée en modifiant l'heuristique JW comme suit :

- $J(x) = H(x) + H(\neg x) + \alpha \times H(x) \times H(\neg x)$
- $H(x) = \sum_{c \in \mathcal{F}, x \in c} \beta_c * 2^{-|c|}$.

L'activité d'un littéral se trouve donc liée à ces occurrences dans les clauses réduites, mais aussi via l'activité de c, à son apparition dans les conflits. Cette heuristique est l'une des premières à associer l'activité d'un littéral à son apparition dans les clauses conflits.

En 2001 dans Zhang *et al.* (2001), les auteurs proposent l'heuristique VSIDS (Variable State Independent Decaying Sum) qui associe à chaque variable un compteur (activité). A chaque fois qu'une clause est violée, toutes les variables de cette clause voient leurs compteurs incrémentés d'un facteur additionnel. A chaque décision, la variable accumulant la meilleure activité est choisie. Cette heuristique dont le coût est dérisoire et facile à mettre à jour est dirigée par les conflits. Les variables apparaissant le plus dans les conflits sont jugées les plus pertinentes.

Une autre généralisation de cette heuristique a été proposée dans le solveur BerkMin Goldberg et Novikov (2002). Le principe est d'augmenter l'activité de toutes les variables rencontrées lors de la génération de la clause assertive. Cette dernière est la plus utilisée actuellement. Reste qu'associer un compteur à chaque variable ne permet pas de décider de la valeur de vérité à choisir lors d'une décision. Pour pallier ce problème, plusieurs heuristiques de polarité ont été proposées dans la littérature parmi lesquelles celle proposée dans Pipatsrisawat et Darwiche (2007) nommée progress saving que nous avons décrit dans le chapitre précédent. Cette heuristique qui est déjà exploité dans le cadre du problème de satisfaction de contrainte (CSPs) a montré son efficacité sur plusieurs classes d'instances de problèmes SAT.

3.7 Heuristiques de Redémarrage

La plupart des solveurs SAT modernes ont une politique de « redémarrage » par laquelle le solveur est forcé de revenir à la prise de niveau 0 selon certains critères. Bien que n'étant pas une technique sophistiquée, il y a des preuves croissantes que cette technique a un impact important sur les performances des solveurs SAT compétitives aujourd'hui.

L'explication courante est que le redémarrage aide le solveur à éviter de passer trop de temps dans les branches où il n'y a ni facilité à trouver une affectation satisfaisante ni les possibilités d'apprentissage rapide des clauses fortes. En effet, dans Gomes *et al.* (2000), les auteurs montrent expérimentalement qu'exécuter la même approche sur le même problème mais avec différents ordres de variables mène souvent à une grande variabilité des temps d'exécution. Ces expérimentations ont permis d'identifier un phénomène singulier nommé phénomène de longue traîne (« heavy tail ») Gent et Walsh (1994), Walsh (1999), Gomes *et al.* (2000), Chen *et al.* (2001). Un redémarage consiste donc à démarrer une nouvelle recherche à priori avec un nouvel ordre de variables. La mojorité des techniques existantes reposent sur un critère global tel que le nombre de conflits appris du redémarrage précédente, et se distinguent par la méthode de calcul du seuil à partir duquel le solveur est obligé de redémarrer. Le parcours de l'espace de recherche change d'un redémarrage à l'autre. Cela est dû soit à l'aspect aléatoire présent dans les algorithmes de recherche, soit au changement de la formule d'entrée augmentée par les clauses ou les deux à la fois.

Dans Ryvchin et Strichman (2008), les auteurs montrent qu'en général la taille des clauses apprises en haut de l'arbre de recherche est en moyenne plus petite qu'en bas de l'arbre.

Nous présentons dans la suite, les deux grandes catégories des stratégies de redémarages les plus utilsées (fonction `redémarrage`). A savoir, les stratégies de redémarrages statiques faisant le plus souvent appel aux séries mathématiques, et les stratégies dynamiques dont le but est d'exploiter différentes informations en cours de recherche afin de déterminer le moment d'effectuer un redémarrage.

3.7.1 Stratégies de redémarrages statiques

Les stratégies de redémarrages statiques ont toutes en commun le fait d'être prédéterminées. Ces politiques reposent le plus souvent sur des séries mathématiques(par exemple : arithmétique, géométrique, Luby). Dans tous ces stratégies, la valeur limite en termes de nombre de conflits avant le redémarrage augmente au fil

du temps. La performance de ces différentes politiques dépend évidemment des instances SAT considérées. En général, il est difficile de dire à l'avance quelle politique devrait être utilisée sur quelle classe de problème Huang (2007)

Nous décrivons ici quelques stratégies de redémarrages statiques utilisées par les Solveurs SAT.

Intervalle fixe : Cette première stratégie toute très naïve consiste à appeler la fonction de redémarrage tous les x conflits. Cette stratégie qui est utilisée dans la version 2004 du solveur ZCHAFF Moskewicz *et al.* (2001) avec un intervalle de $x = 700$ conflits, est également utilisée dans différents solveurs tels que SIEGE Ryan (2004), BERKMIN Goldberg et Novikov (2002) et EUREKA Nadel *et al.* (2006) avec respectivement $x = 16000$, $x = 550$ et $x = 2000$;

Suite géométrique : Le solveur MINISAT 1.13 Eén et Sörenson (2004) est le premier solveur à avoir démontré l'efficacité de la stratégie de redémarrage géométrique suggérée par Walsh (1999). Ici le solveur commence avec un intervalle de redémarrages initial de 100 conflits, qui augmente d'un facteur de 1.5 après chaque redémarrage.

Luby : La série de Luby a été introduite dans Luby *et al.* (1993) et a été utilisée pour la première fois comme stratégie de redémarrage dans le solveur TINISAT Huang (2007). Cette stratégie évolue en fonction d'une série qui est de la forme : $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 4, 8, 1, 1, 2, \ldots$. Puisqu'il est très improbable de trouver une solution après un seul conflit, cette série est multipliée par un facteur. Formellement, l'intervalle entre deux redémarrages t_i et t_{i+1} est effectué après $u \times t_i$ conflits tels que u est une constante représentant le facteur multiplicateur et

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

Cette stratégie a quelques propriétés théoriques intéressantes. En effet, les auteurs montrent dans Luby *et al.* (1993) que cette approche est logarithmiquement optimale lorsqu'aucune information sur le problème n'est fournie. Ainsi à l'heure actuelle elle est devenue le choix des solveurs modernes tels que RSAT 2.0 Pipatsrisawat et Darwiche (2007) et TINISAT Huang (2007) qui l'utilise avec $u = 512$, ainsi que MINISAT 2.1 Sörensson et Eén (2009) et PRECOSAT Biere (2009) qui l'utilise avec $u = 100$;

La figure 3.3 montre l'évaluation de la série de luby avec $u = 512$.

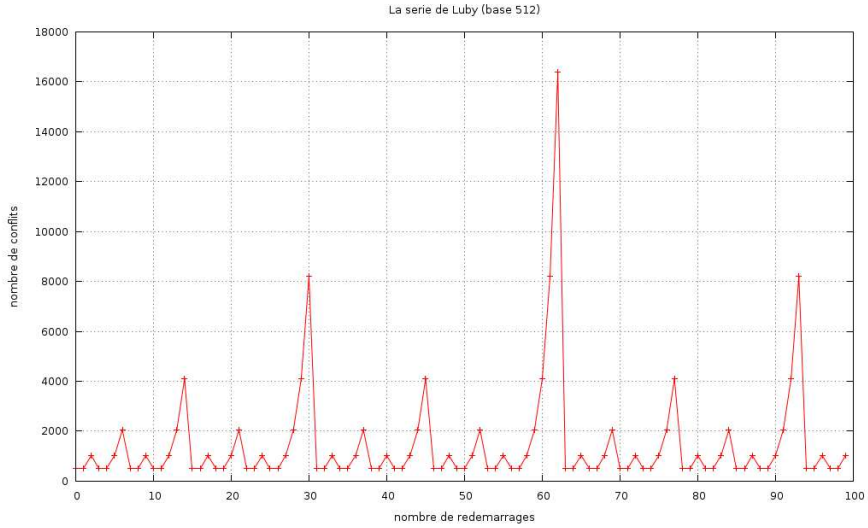


FIGURE 3.3 – La série de luby avec $u = 512$

Inner-outer : Cette stratégie a été implémentée dans le solveur PICOSAT Biere (2008b). Elle est similaire à la stratégie de redémarrage Luby (alternance de redémarrages rapides et lents) et consiste à maintenir une série géométrique extérieure ($outer$) dont le but est de fixer la borne maximale que peut atteindre une série géométrique intérieure ($inner$). Formellement, considérons i la valeur courante de la série intérieure et o la valeur courante de la série extérieure. Lorsque le nombre de conflits obtenus depuis le dernier redémarrage atteint la valeur i, un redémarrage est effectué et i prend la prochaine valeur de la série intérieure. Ensuite, dans le cas où $i > o$ la série intérieure est réinitialisée et o prend la prochaine valeur de la série extérieure.

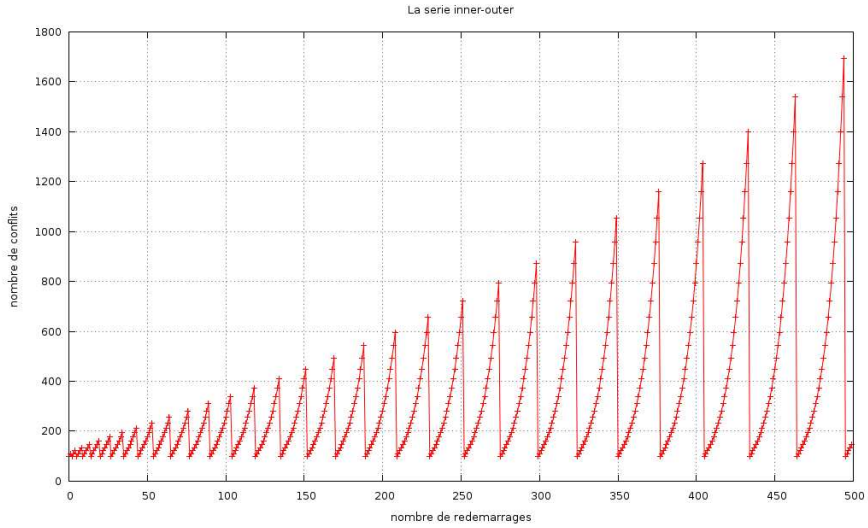


FIGURE 3.4 – La série géométrique inner-outer avec $inner = \{100, 110, \ldots, 100 * 1.1^n\}$,$outer = \{100, 110, \ldots, 100 * 1.1^n\}$

Dans Huang (2007), l'auteur étudie de façon empirique l'impact de toutes ces différentes stratégies de redémarrages sur la comportement des solveurs SAT modernes en comparant les performances des différentes politiques sur un grand nombre d'instances.

3.7.2 Stratégies de redémarrage dynamiques

Depuis quelques années, les stratégies de redémarrages dynamiques qui exploitent les informations issues de la recherche (la taille des résolvantes Pipatsrisawat et Darwiche (2009b), la profondeur moyenne de l'arbre et des sauts arrières Hamadi *et al.* (2009b), le nombre récent de changements de valeurs de variables Biere (2008a)) afin de determiner à quel moment effectuer un redémarrage ont été proposées.

Nous présentons ci-dessous une liste non exhaustive de ces différentes approches :

Variation de la phase :

Cette approche qui est l'une des premières stratégies de redémarrages dynamiques a été proposée par Biere (2008a). Implementée dans le solveur *Picosat*

Biere (2008b), elle a améliorée considérablement ses preformances. Cette technique consiste à mésurer dynamiquement l'agilité du processus de recherche pour contrôler la fréquence de redémarrage. Cette agilité est calculée par l'analyse de l'interprétation complète représentant le *progress saving* (voir §2.5.3). L'agilité est initialisée à zéro ($agility = 0$). Ensuite, lorsqu'une nouvelle variable est affectée, sa nouvelle polarité est étudiée. Si la polarité de la variable est différente de l'ancienne alors l'agilité est augmentée, sinon elle est diminuée. Afin de simuler une forme de « *aging* » sur l'affectation des variables, l'agilité est diminuée par un certain facteur [5] d compris entre 0 et 1 ($agility = agility \times d$). Ce même facteur est aussi utilisé afin d'augmenter l'agilité lors de la mise à jour de celle-ci ($agility = agility + (1 - d)$). De cette manière, il est assuré que la valeur de la variable $agility$ est comprise entre 0 et 1. La politique de redémarrage consiste alors à considérer une stratégie de redémarrage statique utilisant cette notion d'agilité. Lorsque le nombre de conflits à atteint la valeur préconisée par la politique de redémarrage statique, l'agilité courante est comparée avec un certain seuil[5] s et si elle dépasse ce seuil le redémarrage n'est pas effectué. Afin de régler ce seuil, Biere (2008a) émet l'hypothèse qu'une agilité élevée implique que le solveur est susceptible de réfuter le problème et donc qu'il ne faut pas effectuer de redémarrage. Inversement, une agilité basse implique une stagnation et redémarrer la recherche permet d'aider le solveur à s'échapper du sous arbre courant jugé difficile ;

hauteur des sauts : Cette politique de redémarrage proposée par Hamadi *et al.* (2009b) a été implémentée dans le solveur LYSAT. Elle est basée sur l'évolution de la taille moyenne des retours arrières(la moyenne des hauteurs des sauts de chaque retour arrière). L'idée est de délivrer pour de grandes (respectivement petites) variations de la taille moyenne des retours arrières (entre le redémarrage courant et le précédant) une plus petite (respectivement grande) valeur de coupure. Cette fonction est calculée comme suit : $x_1 = 100, x_2 = 100$ et $x_i = y_i \times |cos(1 + r_i)|$, avec $i > 2$, $\alpha = 1200$, y_i représente la moyenne de la taille des retours arrières au redémarrage i, $r_i = \frac{y_{i-1}}{y_i}$ si $y_{i-1} > y_i$ et $r_i = \frac{y_i}{y_{i-1}}$ sinon.

GLUCOSE : la plupart des politiques de redémarrages sont essentiellement basées sur le nombre de conflits afin de déterminer si un redémarrage doit être effectué. Audemard et Simon (2009a) proposent une nouvelle approche qui dépend non seulement du nombre de conflits, mais aussi des niveaux de décision afin de déterminer si un *redémarrage* doit être effectué. Afin d'étudier les différents mécanismes des solveurs SAT modernes, les auteurs ont mené un ensemble d'expérimentations. Une des conclusions de celles-ci est que, sur une grande majorité d'instances, les

5. $d = \frac{1}{10000}$ et $s = \frac{1}{4}$ pour l'analyse expérimentale

niveaux de décisions décroissent tout au long de la recherche et que cette décroissance semble reliée avec l'efficacité (ou inefficacité) des solveurs CDCL Audemard et Simon (2009b). Ainsi la politique de redémarrage proposée par les auteurs essaie de favoriser la décroissance des niveaux de décision durant la recherche : si ce n'est pas le cas, alors un redémarrage est effectué. Pour cela, le solveur GLUCOSE Audemard et Simon (2009a) calcule la moyenne (glissante) des niveaux de décision sur les 100 derniers conflits. Si cette dernière est plus grande que 0.7 fois la moyenne de tous les niveaux de décision depuis le début de la recherche alors un redémarrage est effectué ;

Les différentes approches présentées dans les sections 3.6 et 3.7 montrent que les redémarrages et l'heuristique de choix de variables sont deux composants complémentaires des solveurs SAT modernes dans le sens où ils tendent respectivement à diversifier et à intensifier la recherche. Ainsi le lien entre ces deux composants joue un rôle important dans la résolution du problème SAT. De ce fait, il semble intéressant de trouver une bonne combinaison des deux composants pour l'amélioration de la résolution du problème SAT. Nous proposons au chapitre 5 une nouvelle approche de résolution SAT qui tire parti de ces deux composantes.

3.8 Approches de résolution SAT en parallèle

Les performances des solveurs SAT séquentiels n'ont pas cessé d'augmenter durant ces dernières années, surtout dans la résolution d'instances industrielles dont la taille peut dépasser des centaines de milliers de variables et des millions de clauses. Cette évolution est due non seulement à l'algorithmique bien entendu, mais également à l'augmentation de la puissance de calcul.

En dépit de la puissance de ces solveurs SAT séquentiels, la preuve de la satisfiabilité ou de l'insatisfiabilité de plusieurs problèmes reste un défit intéressant. En effet, de nombreux problèmes industriels restent hors de portés de tous les solveurs contemporains. Par conséquence, de nouvelles approches sont nécessaires pour résoudre ces instances difficiles. Ces récentes années, grâce aux nouvelles architectures multi-cœurs, les approches de résolution parallèle du problème SAT gagnent en intérêt.

Dans cette section, nous présentons brièvement les deux stratégies utilisées dans la littérature pour la résolution parallèle du problème SAT. Cependant, pour plus de détails autour des aspects parallèles de SAT, nous référons les lecteurs à la thèse de Vander-Swalmen (2009).

3.8.1 Approches de type diviser pour régner

Le principe de ce type d'approches consiste à diviser progressivement un problème en plusieurs sous-problèmes et les distribuer aux différents processus. Il existe deux techniques basées sur le principe de division. La première consiste à diviser l'instance originale. Elle peut s'avérer très utile surtout sur les instances de très grande taille. La deuxième vise à décomposer l'espace de recherche.

Les figures 3.5 et 3.6 montrent la différence entre ces deux types de division.

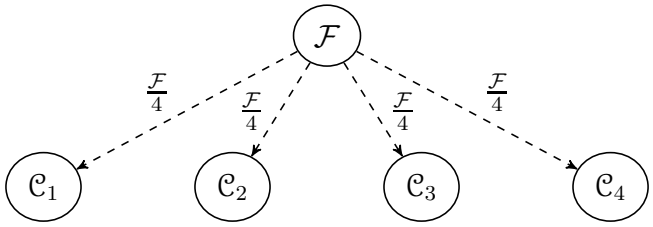


FIGURE 3.5 – Présentation schématique de la division de l'instance. Les cœurs représentent les solveurs séquentiels. Ils prennent en charge chacun une partie de la formule (ici un quart).

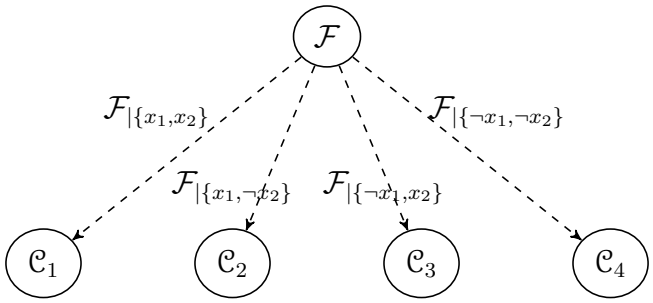


FIGURE 3.6 – Présentation schématique de la division de l'espace de recherche. Chaque cœur représente un solveur séquentiel. Ils prennent en charge chacun la formule avec un chemin de guidage.

Une décomposition de l'espace de recherche est faite à l'aide d'une notion appelée chemin de guidage.

Définition 3.9 (chemin de guidage)**.** *Un chemin de guidage est représenté par un ensemble de couples* $\{\langle l_1, \lambda_1\rangle, \langle l_2, \lambda_2\rangle, \ldots \langle l_n, \lambda_n\rangle\}$*, où* l_i *est un littéral à propager et* λ_i *indique si l'un des sous-arbres issus de* l_i *n'est ni en cours de traitement ni traité. Chaque* λ_i *peut être représenté par une variable booléenne qui est égal à* $\bot$ *si les deux sous-arbres sont traités (ou en cours de traitement) et* $\top$ *sinon.*

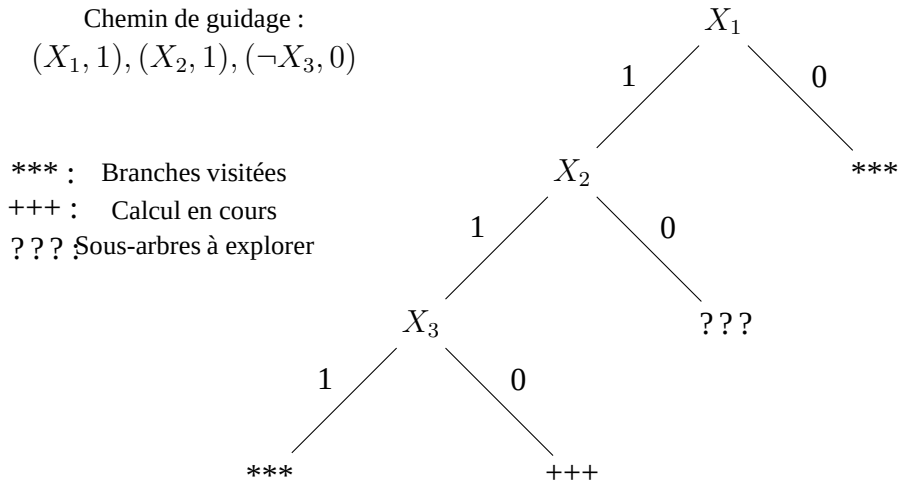


FIGURE 3.7 – Un exemple de chemin de guidage

Une représentation du chemin de guidage est donnée dans la figure 3.7.

Les chemins de guidages sont utilisés pour connaître les sous-arbres qui doivent être explorés, mais aussi assigner du travail aux processeurs. La résolution d'un problème à l'aide de cette approche consiste alors à résoudre l'ensemble des chemins de guidages. Pour cela, lorsqu'une unité de calcul est oisive, un chemin de guidage lui est assigné par le biais d'un des modèles d'équilibrage de charges (redistribution des tâches aux différentes unités de calcul afin de minimiser les temps où les unités de calcul sont oisives). Lorsque l'ensemble des chemins de guidage a été exploré le solveur peut conclure sur la satisfiabilité de la formule.

3.8.2 Approche de type portfolio

Un portfolio d'algorithmes définit une exécution concurrente de plusieurs algorithmes résolvant un même problème.

Les approches de type portfolio consistent à mettre en concurrence sur un même problème plusieurs solveurs séquentiels ayant chacun sa propre stratégie de recherche. Dans une telle exécution, les algorithmes sont entrelacées dans le temps et/ou l'espace. Sur une instance à résoudre, l'exécution est interrompue dès qu'un des solveurs trouve une solution. Le choix des stratégies à assigner aux différents solveurs est une question intéressante. Elle vise essentiellement à ce que l'ensemble des stratégies soient complémentaires et orthogonales.

Certaines approches permettent l'échange d'informations entre les différentes

unités de calculs. Ces informations représentent le plus souvent des clauses obtenues par analyse de conflits et permettent d'améliorer les performances du système au-delà de la performance de chaque solveur considéré individuellement. Cette approche est à l'heure actuelle la meilleure manière de résoudre le problème SAT en parallèle.

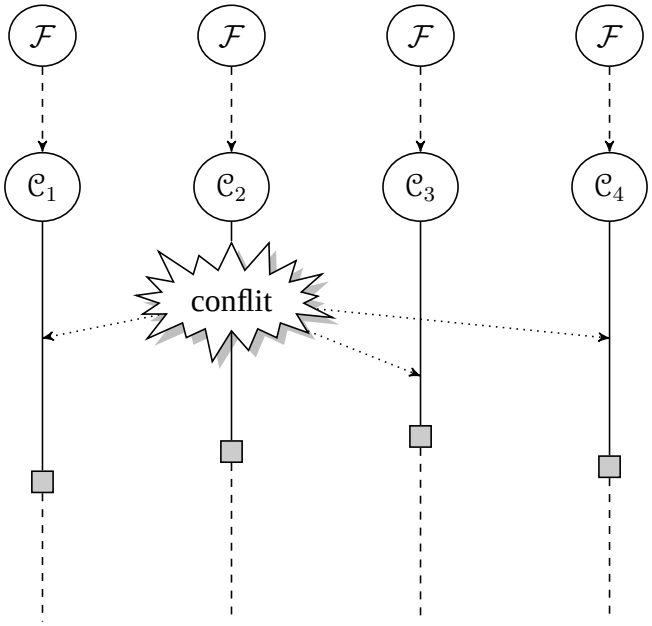


FIGURE 3.8 – Présentation schématique du solveur parallèle de type portfolio. Chaque cœur représente un solveur séquentiel (les stratégies de recherche sont différenciées). Ils prennent en charge chacun la formule originale. Les lignes continues représentent l'exécution d'un solveur. Les carrés symbolisent les redémarrages tandis que les flèches en pointillés représentent le partage d'informations entre les différents solveurs (ici les clauses apprises).

3.9 Conclusion

Dans ce chapitre, nous avons présenté de manière succincte les solveurs SAT modernes avec leurs composantes essentielles. Depuis l'apparition de ces solveurs, des améliorations effectuées sur leurs composantes essentielles ont eu un impact significatif sur leur performance. Nous avons vu qu'en général, ces solveurs ne sont que le résultat d'un subtil mélange de plusieurs composantes. Toute modification apportée sur une de leurs composantes fut-elle légère peut modifier considérablement les performances de ces solveurs. En effet, chaque composant a une influence

directe sur la performance du solveur ainsi que sur les interactions entre les différents composants. Nous avons également évoqué la nécessité de trouver de bonnes combinaisons de certains composants lorsqu'ils sont mis ensemble afin de tirer pleinement profit de l'efficacité de chacun des composants. Ces dernières années, des progrès importants ont été réalisés dans les solveurs SAT modernes, cependant, les dernières évaluations et compétitions internationales (SAT Race et SAT competition) montrent qu'une catégorie d'instances reste toujours difficile à résoudre pour les solveurs SAT actuels. Ainsi, ces instances restent hors de portée des approches actuelles. Dans ce contexte, face à l'évolution des architectures des machines, la mise au point de solveurs SAT parallèles supportant ce type d'architectures constitue une voie importante pour la résolution d'instances SAT difficiles. Nous avons de ce fait consacré le dernier paragraphe de ce chapitre à la présentation des deux types d'approches généralement utiliser pour résoudre SAT en parallèle.

Conclusion

Le problème SAT est le problème de décision qui consiste à déterminer si une formule booléenne mise sous forme normale conjonctive admet ou non un modèle. La simplicité de l'énoncé et du langage propositionnel sous-jacent cache une difficulté théorique majeure. SAT est le premier problème à avoir été montré complet pour la classe NP (Non déterministe Polynomial) par Cook (1971) lui donnant ainsi une place centrale en théorie de la complexité et rendant son intérêt pratique encore plus important. On le retrouve en tant que sous-problème ou cas particulier dans de nombreux domaines comme la déduction automatique, le problème de satisfaction de contraintes, la vérification de circuit, bio-informatique, la planification, etc. L'étude fondamentale et pratique de ce problème est essentielle pour la mise en oeuvre de certains mécanismes de déduction efficace. Grâce à l'utilisation des techniques issues de la résolution du problème SAT, l'on a vu un nouvel essor apparaître dans le domaine de la vérification formelle avec l'introduction du « Bounded Model Checking » (BMC).

Les progrès récents obtenus autour de la résolution pratique de ce problème et les nombreuses applications traitées par la technologie SAT ont permis sans aucun doute de conforter son importance. En effet, plusieurs facteurs contribuent au regain d'intérêt pour SAT. Premièrement, les performances des solveurs n'ont cessé de croître Le Berre (2010), permettant de résoudre des problèmes de plus en plus conséquents (des millions de variables et de contraintes). Deuxièmement, des codages propositionnels plus sophisqués sont proposés pour des problèmes réels et permettent de trouver une solution plus rapidement que dans leur codage d'origine.

L'une des avancées majeures de ces dernières décennies dans la résolution pratique du problème SAT réside dans l'apparition des solveurs SAT modernes basés sur l'architecture CDCL (Conflict Driven Clause Learning Silva et Sakallah (1999), Moskewicz *et al.* (2001), Eén et Sörenson (2004) qui sont une combianison efficace de la procédure DPLL et d'une méthode basée sur le principe de résolution. Les solveurs SAT modernes utilisent en plus des composantes supplémentaires comme une heuristique de branchement basée sur l'activité des variables, et une politique de redémarrage. Ces solveurs ont montré une grande efficacité à traiter des instances, principalement industrielles et crafteds, codées sur des millions de variables et de clauses. L'arrivée sur le marché des ordinateurs avec des architectures multicores a ouvert de nouvelles perspectives permettant la mise au point de solveurs SAT parallèles qui exploitent au maximum la puissance de calcul offerte par cette nouvelle technologie.

Au delà de cette efficacité remarquable des approches de résolution SAT sur

des instances industrielles de très grandes tailles, il reste que de nombreuses classes d'instances restent encore hors de porté des approches actuelles. Ainsi, il y a une demande toujours plus croissante en puissance de calcul pour satisfaire des instances toujours plus grandes et plus difficiles.

Les progrès à accomplir restent donc encore énormes. Il faut trouver de nouvelles stratégies de résolution SAT plus puissantes. Dans la deuxième partie, nous présentons les différentes contributions que nous avons apportées pour l'amélioration de la résolution du problème SAT.

Deuxième partie
Nouvelles Méthodes de Résolution du Problème SAT

Chapitre 4

Résolution Étendue par Substitution Dynamique des Fonctions Booléennes

Sommaire

DANS CE CHAPITRE, nous présentons une technique originale de substitution dynamique de fonctions booléennes. Notre technique procède en deux étapes. Dans une phase de prétraitement, nous effectuons une reconnaissance d'un ensemble de fonctions booléennes à partir d'une formule booléenne sous forme normale conjonctive (CNF). Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises en substituant les arguments d'entrée par les arguments de sortie. Ceci conduit à une façon originale d'intégrer une forme restreinte de résolution étendue, l'un des plus puissant système de preuve par résolution. En effet, la plupart des fonctions booléennes extraites correspondent à celles introduites au cours de la phase de transformation vers CNF en appliquant le principe d'extension Tseitin. Substituer les entrées par les sorties peut être vu comme une façon élégante de mimer la résolution étendue avec la manipulation des sous-formules. Des expérimentations préliminaires montrent la faisabilité de notre approche sur certaines classes d'instances SAT issues des récentes compétitions SAT et SAT-Race.

4.1 Introduction

L'apprentissage de clauses est connu comme l'un des éléments les plus importants des solveurs SAT modernes. Comme nous l'avons évoqué dans les chapitres précédents, l'idée principale est que quand une branche courante de l'arbre de recherche conduit à un conflict, l'apprentissage de clauses vise à dériver une clause qui succinctement exprime les causes de ce conflit. Cette clause apprise est ensuite utilisée pour élaguer l'espace de recherche. L'apprentissage de clauses est également connu dans la littérature comme "Conflict Driven Clause Learning" (CDCL) faisant référence au schéma le plus connu et le plus utilisé, appelé *First UIP*. L'apprentissage de clauses a d'abord été intégré dans les solveurs SAT Grasp Marques-Silva et Sakallah (1996b) et mis en oeuvre efficacement dans zChaff Moskewicz *et al.* (2001). Les solveurs SAT modernes peuvent être vus comme une extension de la bien connue procédure DPLL classique obtenue grâce à différentes améliorations. Il est important de noter que la bien connue règle de résolution joue encore un rôle prépondérant dans l'efficacité des solveurs SAT modernes qui peut être vu comme une forme particulière de la résolution générale Beame *et al.* (2003).

Théoriquement, en intégrant l'apprentissage de clauses à la procédure DPLL classique Davis *et al.* (1962), le solveur SAT obtenu formulé comme un système de preuve est aussi puissant que la résolution générale Pipatsrisawat et Darwiche (2009a). Ce résultat théorique important, suggère que, pour améliorer l'efficacité des solveurs SAT, on a besoin de trouver des systèmes de preuves plus puissants. Cette question de recherche est également motivée par l'existence de plus en plus des des instances qui ne peuvent actuellement être résolues par les solveurs SAT disponibles. La résolution étendue Tseitin (1968) et la résolution par symétrie Krishnamurthy et Moll (1981) sont les deux systèmes de preuves qui sont connus comme étant plus puissants que la résolution. Le principe d'extension permet l'introduction des variables auxiliaires pour représenter les formules intermédiaires de telle sorte que la longueur d'une preuve peut être considérablement réduite en manipulant ces variables au lieu de manipuler les formules qu'elles représentent. La symétrie, d'autre part, permet de reconnaître qu'une formule reste invariante sous certaines permutations de noms de variables, et utilise cette information pour éviter de répéter certaines dérivations pouvant être obtenues par symétries.

Pour la symétrie, plusieurs approches ont été proposées à la fois dans la satisfiabilité booléenne (e.g. Crawford *et al.* (1996), Benhamou et Sais (1994), Aloul *et al.* (2003)) et dans la programmation par contraintes (e.g. Puget (1993), Gent et Smith (2000), Flener *et al.* (2009)). Cependant, l'automatisation du système de preuve par résolution étendue reste encore une question ouverte. La résolution étendue ajoute une règle d'extension au système de preuve par résolution, permettant l'introduction les définitions dans la preuve. Par exemple étant donnée une formule booléenne $\mathcal{F}$,

contenant les variables a et b, et v une nouvelle variable, on peut ajouter la définiton $v \leftrightarrow a \vee b$ tout en préservant la satisfiabilité. Rappelons que la règle d'extension est à la base de la transformation linéaire des formules booléennes générales en forme normale conjonctive (CNF). Par exemple, la formule $\mathcal{F} = (x_1 \leftrightarrow x_2 \leftrightarrow \cdots \leftrightarrow x_n)$ ne peut être transformée polynomialement en une formule $CNF(\mathcal{F})$ équivalente. Cependant, en utilisant le principe d'extension, on peut obtenir une transformation linéaire de $\mathcal{F}$ à $CNF(\mathcal{F})$. La formule CNF obtenue est équivalente à celle d'origine par rapport à la satisfiabilité. Il est également démontré que certaines formules qui sont difficiles pour la résolution (e.g. Pigeon-hole formulas Haken (1985)) admettent des courtes preuves par résolution étendue Cook (1976). La preuve courte pour le problème des pigeons est obtenue en simulant le principe d'induction par l'utilisation de la règle d'extension. Comme mentionné par B. Krishnamurthy dans Krishnamurthy (1985), l'extension permet la définition d'hypothèses intermédiaires qui peuvent être introduites par le principe d'extension. En manipulant ces hypothèses, de courtes preuves peuvent être obtenues pour certaines formules difficiles.

Récemment, un solveur SAT CDCL basé sur une restriction de la règle d'extension a été proposé dans Audemard *et al.* (2010) . Plus précisement, quand 2 clauses successives apprises par le solveur sont de la forme $l_1 \vee \alpha$, $l_2 \vee \alpha$, une nouvelle variable $z \leftrightarrow \neg l_1 \vee \neg l_2)$ est introduite.

Comme nous l'avons souligné au chapitre 2, la principale difficulté derrière l'automatisation de la résolution étendue est de déterminer (1) quand est ce que de telles définitions seront ajoutées et (2) quelles fonctions booléennes doivent-elles représenter. L'approche proposée dans ce chapitre exploite les nouvelles variables introduites dans la phase d'encodage des formules booléennes générales en CNF grâce à l'application du principe d'extension de Tseitin. Substituer ces variables pendant la recherche peut être vu comme une façon élégante de manipuler dynamiquement ces sous-formules. Cela imite clairement le principe de la résolution étendue. Notre proposition répond à la fois aux questions (1) et (2), en exploitant les fonctions booléennes cachées généralement introduites lors de l'encodage, et en les utilisant afin de minimiser les clauses apprises. De plus, comme on peut le voir plus tard, notre approche à une certaine similitude et différence avec la simplification par hyper-résolution binaire Bacchus et Winter (2003), Bacchus (2002).

Ce chapitre est organisé comme suit. Dans la première section, nous rappelons la notion de fonctions booléennes, Après avoir présenté quelques techniques de detection de ces fonctions booléennes dans une formule CNF (section 4.3), notre approche de substitution dynamique des fonctions booléennes est décrite dans la section 6.3. Finalement, avant de conclure, les résultats expérimentaux démontrant la faisabilité de notre approche sont présentés.

Les travaux présentés dans ce chapitre ont fait l'objet de plusieurs publications :

Jabbour *et al.* (2012d;a; 2014b).

4.2 Fonctions booléennes

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

- $y = \wedge(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{y \vee \neg x_1 \vee \cdots \vee \neg x_n, \neg y \vee x_1, \ldots, \neg y \vee x_n\}$, traduisant la condition selon laquelle la valeur de vérité de y est determinée par la conjonction des valeurs de vérité des x_i pour $i \in [1 \ldots n]$;
- $y = \vee(x_1, \ldots, x_n)$ représente l'ensemble de clauses $\{\neg y \vee x_1 \vee \cdots \vee x_n, y \vee \neg x_1, \ldots, y \vee \neg x_n\}$;
- $y \Leftrightarrow (x_1, \ldots, x_n)$ représente la chaîne d'équivalences (formule biconditionnelle) $y\{\Leftrightarrow x_1 \Leftrightarrow ... \Leftrightarrow x_n\}$ qui correspond à 2^n clauses

Dans ce travail, nous nous intéressons uniquement aux fonctions booléennes de la forme $y = f(x_1, \ldots, x_n)$, où $f \in \{\vee, \wedge\}$ encodées dans une formule CNF et nous ne considérons que les fonctions booléennes de la forme $y = \vee(x_1, \ldots, x_n)$ où y et x_i sont des variables booléennes de la formule originale. En effet, une fonction booléenne $y = \wedge(x_1, \ldots, x_n)$ peut être écrite comme $\neg y = \vee(\neg x_1, \ldots, \neg x_n)$

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

4.3 Détection des fonctions booléennes dans une formule CNF

Etant donné une formule CNF $\mathcal{F}$, On peut utiliser deux méthodes différentes pour la détection des fonctions booléennes encodées par les clauses de $\mathcal{F}$.

4.3.1 méthode syntaxique (pattern matching)

La première méthode de détection, appelée méthode syntaxique, est une approche de type "pattern matching" qui nous permet de detecter les fonctions booléennes qui apparaissent directement dans la structure de la formule CNF Ostrowski *et al.* (2002).

Exemple 4.1. *Soit* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg y \vee x_2), (\neg y \vee x_3)\}$.

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

4.3.2 méthode sémantique

La seconde méthode est l'approche de détection sémantique où les fonctions sont détectées en utilisant la propagation unitaire (PU) Grégoire *et al.* (2005). En effet, cette méthode permet de détecter les fonctions booléennes cachées.

Exemple 4.2. *Soit* $\mathcal{F} \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg x_1 \vee x_4), (\neg x_4 \vee x_2), (\neg x_2 \vee x_5), (\neg x_4 \vee \neg x_5 \vee x_3)\}$.

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

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

Ces fonctions sont ensuite utilisées pour réduire la taille des différentes clauses apprises durant la recherche. Dans notre implémentation, nous exploitons les deux approches de détection des fonctions booléennes proposées dans Ostrowski *et al.* (2002) et Grégoire *et al.* (2005).

4.4 Substitution Dynamique des Fonctions Booléennes

Dans cette section, nous montrons comment les fonctions booléennes détectées dans la formule originale peuvent être utilisées pour réduire dynamiquement la taille des différentes clauses apprises.

4.4.1 Motivation

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

Exemple 4.3. *Soit $\mathcal{F}$ la formule CNF précédente, supposons que $\mathcal{F}$ contient aussi les clauses suivantes $\{c_{13}, \ldots, c_{16}\}$ i.e. $\{c_{13}, \ldots, c_{16}\} \supseteq \mathcal{F}$*

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

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

Exemple 4.4. *Soit $\mathcal{F}$ la formule CNF précédente, supposons maintenant que $\mathcal{F}$ contient plutôt les clauses suivantes $\{c_{13}, \ldots, c_{17}\}$ i.e. $\{c_{13}, \ldots, c_{17}\} \supseteq \mathcal{F}$*

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

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

La méthode de détection basée sur la propagation unitaire nous permet de détecter toutes les fonctions booléennes que nous ne pouvons pas détecter syntaxiquement. Il est important de noter que lorsque les fonctions booléennes sont détectées en utilisant la propagation unitaire, la résolvante σ''_7 ne peut pas être obtenue directement par hyper-binaire résolution. Pour que cela soit possible, nous avons besoin

premièrement de générer les clauses $(\neg y \vee x_{12})$ et $(\neg y \vee x_6)$ et par résolution avec les clauses c_{14}, c_{15}, c_{16} et c_{17}.

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

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

4.4.2 Substitution Dynamique

Nous donnons à présent une présentation formelle de notre approche basée sur la substitution dynamique.

Proposition 4.1. *[Substitution dynamique] Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, avec x comme littéral assertif et α une disjonction de littéraux. Soit $y = \vee(x_1, \ldots, x_n)$ avec $n \geq 2$, une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q. $\{x_1, \ldots, x_n\} \subseteq \alpha$, alors σ peut être réduite à σ' en substituant $\{x_1, \ldots, x_n\}$ par y dans α t.q. :*

- *$\sigma' = \beta \vee x$ si $y \in \alpha$*
- *$\sigma' = \beta \vee x \vee y$ sinon*

  avec $\beta = \alpha \backslash \{x_1, \ldots, x_n\}$

Propriété 4.2. *Soit $\mathcal{F}$ une formule CNF, $\sigma = (\alpha \vee x)$ une clause apprise, où x est le littéral assertif. Soit $y = \vee(x_1, \ldots, x_n)$ une fonction booléenne encodée par les clauses de $\mathcal{F}$ t.q. $\{x_1, \ldots, x_n\} \subseteq \alpha$, on a :*

- *$\forall 1 \leq i \leq n, \rho(x_i) = faux$ et $\rho(y) = faux$*
- *$l(y) = max\{l(x_i), 1 \leq i \leq n\}$ et $l(y) < l(x)$*

Preuve 4.1. *Par définition de la clause apprise σ, on a $\rho(\sigma) = faux$. Comme $\{x_1, \ldots, x_n\} \subseteq \sigma$, par conséquent, $\forall 1 \leq i \leq n, \rho(x_i) = faux$.*
On a $y = \vee(x_1, \ldots, x_n)$ alors $\rho(y) = faux$.
Par définition de la fonction booléenne $y = \vee(x_1, \ldots, x_n)$, $l(y)$ est le niveau du dernier littéral x_i qui fut affecté à faux. Par conséquent, $l(y) = max\{l(x_i), 1 \leq i \leq n\}$. Comme x est le littéral assertif, par définition, $\forall z \in \alpha, l(z) < l(x)$. Puisque $\{x_1, \ldots, x_n\} \subseteq \alpha$ et $l(y) = max\{l(x_i), 1 \leq i \leq n\}$, alors $l(y) < l(x)$.

Propriété 4.3. *Soit $\sigma = (\alpha \vee x)$ une clause apprise, et $G = \{g_1, \ldots, g_n\}$ un ensemble de fonctions booléennes détectées dans la formule originale telles que les différents ensembles de variables d'entrée associées aux différentes fonctions booléennes dans G soient tous disjoints et tous inclus dans α. Soit $\sigma_1, \ldots, \sigma_n$ la séquence des clauses apprises obtenues successivement après chaque substitution tel que à l'étape i, σ_i est obtenue en remplaçant un ensemble de variables d'entrée dans σ_{i-1} par une variable de sortie y_j, alors $|\sigma_i| < |\sigma_{i-1}|$, et $|\sigma_i| > 1$, $1 \leq i, j \leq n$.*

Preuve 4.2. *Soit σ_0 la clause initialement apprise. Comme σ_i est obtenue par substitution d'au moins deux littéraux (entrée d'une fonction booléenne) dans σ_{i-1} par un littéral qui est la sortie de cette fonction, on a $|\sigma_i| < |\sigma_{i-1}|$, pour $1 \leq i \leq n$. De plus, $\forall 1 < i < n$, $x \in \sigma_i$, où x est le littéral assertif et n'est pas considéré dans la substitution, alors $|\sigma_i| > 1$, $1 \leq i \leq n$.*

Propriété 4.4. *Etant donné une clause apprise σ, et un ensemble $\mathcal{G} = \{g_1, \ldots, g_n\}$ de fonctions booléennes détectées dans la formule CNF originale. La complexité temporelle de notre approche de substitution dynamique est quadratique dans le pire des cas.*

Preuve 4.3. *On a $|\mathcal{G}|$ fonctions booléennes. Pour chaque fonction booléenne, on effectue $|\sigma|$ comparaisons. Ainsi, on peut faire au plus $|\mathcal{G}| \times |\sigma|$ comparaisons pour réaliser la substitution.*

4.5 Expérimentations

Les expérimentations menées sont effectuées sur un large panel d'instances industrielles et crafted provenant de la compétition SAT 2009 et de la SAT-Race 2010. Toutes les instances sont simplifiées par pré-traitement `SatElite` Eén et Biere (2005). On peut noter que `SatElite` substitue certaines fonctions booléennes à l'étape de pré-traitement. Par conséquent, les fonctions booléennes détectées sont celles qui restent dans la formule simplifiée par `SatElite`. Nous avons intégré notre approche de substitution dynamique dans MINISAT 2.2 Eén et Sörenson (2004) et nous avons effectué une comparaison entre les résultats obtenus par le solveur original et celui incluant l'approche de substitution dynamique. Il est aussi important de noter le détail d'implémentation suivant : Quand les variables d'entrée sont substituées par la variable de sortie, l'activité de la variable de sortie est mise à jour de la même manière que les autres variables de la clause apprise. Ceci permet de focaliser la recherche sur certaines variables de sortie. Ce détail d'implémentation est similaire à celui implémenté dans Audemard *et al.* (2010).

Les tests sont effectués sur un cluster Xeon 3.2GHz (32 GB RAM). Les résultats du temps de calcul sont indiqués en secondes.

Pour ces expérimentations, le temps de calcul limite est fixé à 1 heure. les différentes tables ci-dessous montrent un ensemble représentatif de familles d'instances.

4.5.1 Problèmes industriels

Les tables 4.1 et 4.2 détaillent les résultats sur les problèmes SAT industriels issus de la compétition SAT'2009 et SAT-Race'2010 en utilisant respectivement les méthodes de détection syntaxique et sémantique pour la détection des fonctions booléennes dans la formule originale.

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

La table 4.1 représente un focus sur quelques familles industrielles. Sur ces familles, les améliorations sont relativement importantes. par exemple, si on considère la famille vmpc, on peut voir que notre substitution dynamique permet d'avoir un gain d'un ordre de magnitude (instances 24 and 27). Sur la même famille, `Minisat+DS` résoud 2 instances de plus que `Minisat` (instances 31 and 33).
Sur la famille gss, sur les 7 instances résolus par `Minisat+DS` et `Minisat`, `Minisat+DS` est meilleur sur 5 instances. Sur la même famille, `Minisat+DS` résoud une instance de plus que `Minisat` (instance 23).

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

4.5.2 Problèmes crafteds

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

La table 4.3 montre que sur les 11 instances résolues de la famillle QG, `Minisat+DS` résoud plus efficacement 7 instances. Sur la famille rbsat, `Minisat` est meilleur sur 5 instances parmi les 8 instances résolues. On peut remarquer que sur 4 de ces 5 instances, la valeur de $nbSum$ et $totalSub$ est égal à 0 ce qui peut expliquer les limites de l'efficacité de notre approche dans ces cas.

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

La table 4.4 montre que ces performances sont encore améliorées en utilisant la méthode sémantique pour la détection des fonctions boolénnes. Dans cette table, sur la famille QG, les temps de résolution de `Minisat+DS` ont été améliorés de manière significative.

4.5.3 Résumé des expérimentations

Dans l'ensemble, nos expérimentations nous ont permis de démontrer deux choses. Premièrement, notre technique ne dégrade pas mais au contraire améliore souvent les performances du solveur original sur les problèmes industriels et crafted. Deuxièmement, elle améliore l'efficacité de ces algorithmes sur des classes de problèmes combinatoires difficiles (crafted). Les résultats présentés dans les tables précédentes montre que notre approche est éfficace sur certaines familles d'instances SAT. Globalement, sur l'ensemble d'instances, notre approche est compétitive et résoud un nombre similaire d'instances.

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

4.6 Conclusion

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

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

TABLE 4.1 – Instances Industrielles : substitution dynamique en utilisant la méthode de détection syntaxique

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

TABLE 4.2 – Instances Industrielles : substitution dynamique en utilisant la méthode de détection sémantique

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

TABLE 4.3 – Instances Crafted : substitution dynamique en utilisant la méthode de détection syntaxique

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

TABLE 4.4 – Instances Crafted : substitution dynamique en utilisant la méthode de détection sémantique

complémentarité. De plus, notre approche présente des améliorations intéressantes sur certaines familles d'instances SAT.

Chapitre 5

Intensification de la Recherche dans les Solveurs SAT Modernes

Sommaire

DANS CE CHAPITRE, nous proposons une nouvelle approche de recherche basée sur le principe d'intensification. En effet, les redémarrage et la recherche basée sur les activités sont deux composantes importantes et liées des solveurs SAT Modernes. D'une part, la mise à jour des activités des variables impliquées dans l'analyse des conflits vise à circonscrire la partie la plus importante de la formule booléenne. Alors que le redémarrage permet au solveur de réorganiser les variables en focalisant la recherche sur la partie la plus importante de l'espace de recherche. Cette combinaison permet au solveur de diriger la recherche vers la sous-formule la plus contraignante. Dans ce chapitre, nous proposons de mettre en avant cette recherche basée sur l'intensification en collectant les variables rencontrées au cours de la dernière analyse de conflit. A chaque redémarrage, ces variables sont à nouveau sélectionnées en utilisant l'ordre prédéfini. Ce simple principe d'intensification apporte des améliorations significatives lorsqu'il est intégré aux solveurs SAT modernes.

5.1 Introduction

Le redémarrage qui est devenu un composant essentiel des solveurs SAT modernes depuis les travaux de Gomes *et al.* (2000) et la recherche basée sur les activités sont deux composants importants et liés des Solveurs SAT Modernes. D'une part, la mise à jour des activités des variables impliquées dans l'analyse des conflits

vise à circonscrire la partie la plus importante de la formule booléenne. Alors que le redémarage permet au solveur de réorganiser les variables en focalisant la recherche sur cette sous-formule contraignante et de préserver le solveur des phénomènes de longue traîne *"heavy tailed"* Gomes *et al.* (1997; 2000). Cette combinaison permet au solveur d'intensifier la recherche et en même temps d'éviter le trashing, c'est à dire de visiter le même sous espace plusieurs fois. Cette forte connexion bien connue entre les redémarrages et l'ordonnancement des variables a aussi une conséquence directe sur l'apprentissage de clauses. Les effets de redémarrage sur la clause apprise ont été largement étudiés (e.g. Biere (2008a), Huang (2007), Pipatsrisawat et Darwiche (2009b)). Notre intuition est que si les solveurs SAT sont capables de résoudre efficacement des instances applicatives avec des millions de variables et de clauses, cela signifie que la partie la plus contraignante de la formule (ou sous-ensemble de variables) est de taille raisonnable. Ceci est lié à l'observation faite précédemment sur la taille des ensembles backdoor Williams *et al.* (2003b;a) observé dans plusieurs domaines d'applications.

Dans les travaux précédents, et dans le solveur parallèle porfolio Hamadi *et al.* (2009c), les auteurs ont montré comment les deux principes bien connus de diversification et d'intensification peuvent être combinés dans le contexte de l'architecture Maîtres/Esclaves Guo *et al.* (2010). Les Maîtres exécutent une stratégie de recherche originale, en veillant à la diversification, tandis que les unités restantes, classées comme des esclaves sont là pour intensifier la stratégie de leur maître. L'intensification consiste à forcer l'esclave à explorer différemment autour de l'espace de recherche exploré par le maître.

Dans ce chapitre, nous proposons de mettre en avant cette recherche basée sur l'intensification en collectant les variables rencontrées au cours de la dernière analyse de conflit. Ainsi, à chaque redémarrage, le solveur branche en priorité sur ces variables. Ce simple principe d'intensification apporte des améliorations significatives lorsqu'il est integré au solveur SAT Minisat2.2.

Le chapitre est organisé de la façon suivante. Nous décrivons tout d'abord notre approche d'intensification dans la section suivante. Ensuite, avant de conclure, nous présentons les résultats expérimentaux démontrant l'efficacité de cette approche.

Les travaux présentés dans ce chapitre ont donné lieu à deux publications Jabbour *et al.* (2012b;c).

5.2 Intensification de la Recherche

Comme nous avons mentionné dans la section précédente, le lien entre les redémarrages et les heuristiques d'ordonnancement des variables joue un rôle important

dans la résolution SAT. Ainsi, l'idée de base de ce chapitre est focalisée sur les relations entre ces deux composants. En effet, au cours de la recherche, à chaque conflit, un parcours du graphe d'implications est effectué à partir du conflit jusqu'au dernier UIP (nœud x_{11} - coupe 3 dans la Figure 5.1) et l'ensemble des variables correspondant aux différents nœuds visités sont insérées dans une file. A chaque redémarrage, le solveur branche en priorité sur les variables collectées. Lorsque toutes les variables de la file sont affectées, le solveur continue avec l'heuristique de branchement ordinaire VSIDS Moskewicz *et al.* (2001). De cette façon, les variables les plus proches du côté des conflits sont d'abord affectées en utilisant les polarités des littéraux progressivement sauvegardées Pipatsrisawat et Darwiche (2007).

5.2.1 Intensification de la Recherche : Exemple illustratif

Nous illustrons cette nouvelle approche de recherche basée sur l'intensification en utilisant un simple exemple. Soit $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}}$ (Figure 5.1) un graphe d'implications associé à la formule CNF $\mathcal{F}$ et une interprétation partielle $\mathcal{I}$. Chaque nœud dans le graphe d'implications correspond à une affectation d'un littéral de décision (nœuds x_4, x_{17} et x_{11} affectés respectivement au niveau 2, 3 et 5) ou à un littéral assigné par propagation unitaire. Par exemple le littéral x_{16} est propagé grâce à la clause c_2 au niveau 5. Nous supposons que le dernier conflit rencontré juste avant le redémarrage est représenté par le graphe de conflit suivant.

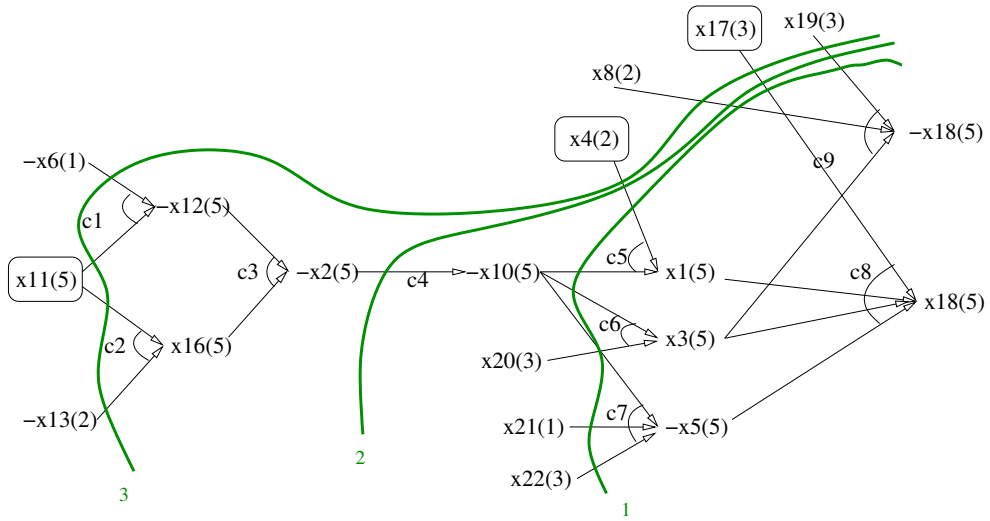


FIGURE 5.1 – Graphe d'Implications $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}} = (\mathcal{N}, \mathcal{A})$

L'analyse de conflit qui est l'application de la règle de résolution à partir de la clause conflictuelle en utilisant les différentes implications implicitement encodées

dans le graphe d'implications nous permet de visiter le nœud x_{11} qui est le dernier UIP (le littéral de décision affecté au niveau du conflit). Au cours de cette analyse, l'ensemble des différents nœuds visités est $\{ x_{18}, \neg x_5, x_3, x_1, \neg x_{10}, \neg x_2, \neg x_{12}, x_{16}, x_{11}\}$. En effet, Ces différents nœuds visités réprésentent les différentes variables des étapes de résolution au cours du processus d'analyse de conflit. Plus précisement, si nous considérons le processus d'analyse de conflit basée sur le graphe de la figure 5.1, les différentes variables sont collectées au cours des différentes étapes de résolution comme suit :

- $\sigma_1 = \eta[x_{18}, c_8, c_9] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_1}^5 \vee \neg {x_3}^5 \vee {x_5}^5)$
  La dérivation de σ_1 nous permet de visiter x_{18}.
- $\sigma_2 = \eta[\neg x_5, \sigma_1, c_7] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \neg {x_1}^5 \vee {x_{10}}^5 \vee \neg {x_3}^5)$
  La dérivation de σ_2 nous permet de visiter $\neg x_5$.
- $\sigma_3 = \eta[x_3, \sigma_2, c_6] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee x_{10}^5 \vee \neg {x_1}^5)$
  La dérivation de σ_3 nous permet de visiter x_3.
- $\sigma_4 = \eta[x_1, \sigma_3, c_5] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee \neg {x_4}^2 \vee {x_{10}}^5)$
  La dérivation de σ_4 nous permet de visiter x_1.
- $\sigma_5 = \eta[\neg x_{10}, \sigma_4, c_4] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee \neg {x_4}^2 \vee {x_2}^5)$
  La dérivation de σ_5 nous permet de visiter $\neg x_{10}$.
- $\sigma_6 = \eta[\neg x_2, \sigma_5, c_3] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee \neg {x_4}^2 \vee \neg {x_{16}}^5 \vee {x_{12}}^5)$
  La dérivation de σ_6 nous permet de visiter $\neg x_2$.
- $\sigma_7 = \eta[\neg x_{12}, \sigma_6, c_1] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee \neg {x_4}^2 \vee {x_6}^1 \vee \neg {x_{11}}^5 \vee \neg {x_{16}}^5)$
  La dérivation de σ_7 nous permet de visiter $\neg x_{12}$.
- $\sigma_8 = \eta[x_{16}, \sigma_7, c_2] = (\neg {x_{19}}^3 \vee \neg {x_8}^2 \vee \neg {x_{17}}^3 \vee \neg {x_{21}}^1 \vee \neg {x_{22}}^3 \vee \neg {x_{20}}^3 \vee \neg {x_4}^2 \vee {x_6}^1 \vee {x_{13}}^2 \vee \neg {x_{11}}^5)$
  La dérivation de σ_8 nous permet de visiter x_{16}.

$\neg x_{11}$ correspond au dernier UIP puisque x_{11} est la dernière variable de décision. Cette dernière variable de décision est également insérée dans la file. Cet ensemble de variables est ainsi collecté dans la file en suivant les différentes étapes de résolution. Au prochain redémarrage, le solveur affecte en priorité les variables de cet ensemble.

5.2.2 Intensification de la Recherche : Formulation générale

Soit $C = \{y_1, y_2, \ldots, y_k\}$ l'ensemble des littéraux rencontrés pendant l'analyse du dernier conflit survenu juste avant le redémarrage du solveur. Plus précisement, les littéraux dans C correspondent aux nœuds situés entre le conflit et le dernier UIP dans le graphe d'implications, où y_1 représente le littéral du conflit et y_k correspond au dernier UIP. Au redémarrage, nous dirigeons la recherche autour de ce dernier conflit en imposant au solveur de brancher en priorité sur les variables de l'ensemble C. Ces variables sont selectionnées en suivant l'ordre dans lequel les différents nœuds correspondant dans le graphe d'implication ont été visités. Elles sont affectées en utilisant les polarités des littéraux $y_i, 1 < i < k$ progressivement sauvegardées Pipatsrisawat et Darwiche (2007). Lorsque toutes les variables de C sont affectées, le solveur continue avec l'heuristique de branchement ordinaire VSIDS Moskewicz *et al.* (2001).

5.3 Expérimentations

Nous présentons dans cette section les résultats expérimentaux obtenus en intégrant notre principe d'intensification au sein du solveur MINISAT Eén et Sörenson (2004). Nos tests furent effectués sur un cluster Intel Xeon quadcore avec 32GB de RAM à 2.66 Ghz. Pour chaque instance, nous utilisons un temps limite d'une heure. Nous utilisons un ensemble d'instances industrielles pris des compétitions SAT 2009 et 2011. Le nombre d'instances différentes correspond à 559.

MINISAT avec notre stratégie ($MiniSat+Intensification$) résoud 12 instances de plus que MINISAT sans intensification. Le nombre d'instances résolues en plus est clairement significatif dans la résolution pratique de SAT. En effet, si on observe la compétition SAT 2011, le solveur classé premier résoud seulement 4 instances de plus que le solveur classé deuxième.

Ces résultats obtenus par notre simple recherche basée sur l'intensification sont représentés dans les figures 5.2 et 5.3. Ils représentent les résultats des temps cumulés i.e le nombre d'instances $(axe - x)$ résolues en un temps donné en secondes $(axe - y)$. La figure 5.2 (respectivement figure 5.3) montre les résultats obtenus sur les instances satisifiables (respectivement insatisfiables). Les améliorations sont souvent plus importantes sur les instances insatisfiables. A partir de ces résultats, nous pouvons observer que $MiniSat + Intensification$ est généralement plus rapide et résoud plus d'instances que MINISAT sans intensification. Ceci confirme l'efficacité de notre approche.

La table 5.1 montre les résultats sur certaines familles d'instances SAT. Cette table montre des améliorations consistantes presque sur toutes les instances quand

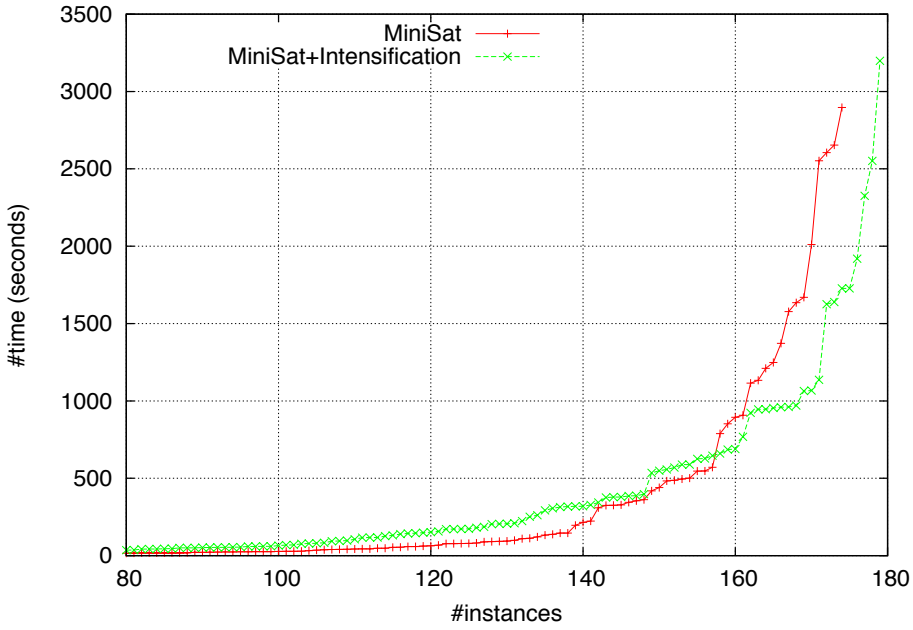


FIGURE 5.2 – Résultats sur les compétitions SAT 2009 et 2011 - Instances Satisfiables - (Catégorie Applications)

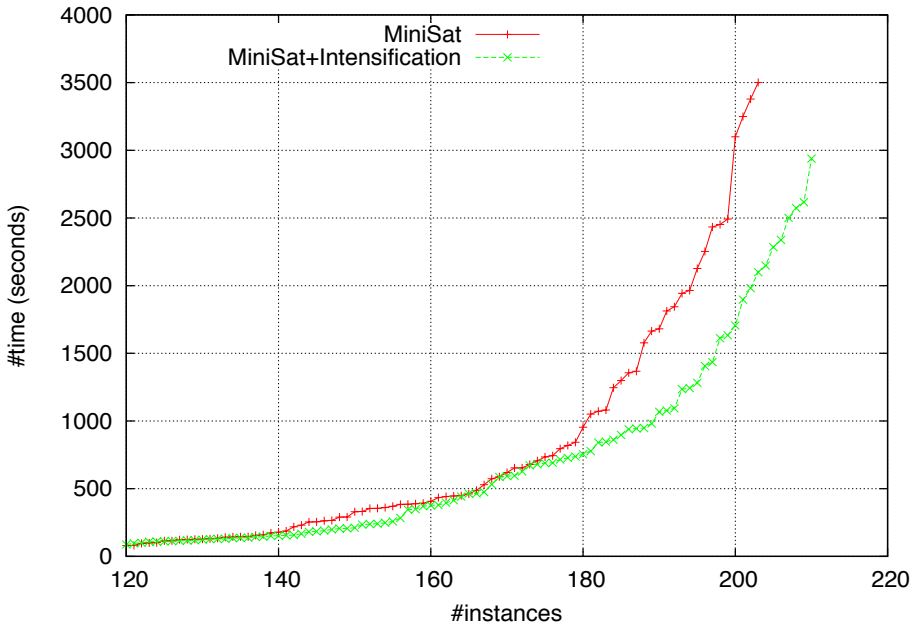


FIGURE 5.3 – Résultats sur les compétitions SAT 2009 et 2011 - Instances Insatisfiables - (Catégorie Applications)

Instance	SAT ?	MiniSat	MiniSat+Intensification
goldb-heqc-alu4mul	N	140.94	128.64
goldb-heqc-term1mul	N	54.79	38.40
goldb-heqc-i10mul	N	187.06	129.14
goldb-heqc-dalumul	N	1663.95	181.31
goldb-heqc-frg1mul	N	–	715.66
goldb-heqc-x1mul	N	–	2500.59
velev-engi-uns-1.0-4nd	N	9.12	14.60
velev-live-uns-2.0-ebuf	N	18.53	9.84
velev-pipe-sat-1.0-b7	Y	21.51	23.51
velev-vliw-uns-4.0-9C1	N	3378.39	112.87
velev-pipe-o-uns-1.1-6	N	619.57	28.26
velev-pipe-o-uns-1.0-7	N	446.30	441.81
9dlx_vliw_at_b_iq1	N	1964.90	28.86
9dlx_vliw_at_b_iq2	N	–	70.07
9dlx_vliw_at_b_iq7	N	2642.42	1282.10
9dlx_vliw_at_b_iq3	N	–	189.83
9dlx_vliw_at_b_iq4	N	1750.56	283.74
9dlx_vliw_at_b_iq8	N	2293.63	1633.04
9dlx_vliw_at_b_iq9	N	2410.76	2572.97
9dlx_vliw_at_b_iq5	N	1631.90	464.47
9dlx_vliw_at_b_iq6	N	1267.90	840.68
velev-pipe-uns-1.0-8	N	–	414.21
velev-vliw-uns-4.0-9-i1	N	2095.14	592.52
velev-pipe-sat-1.0-b10	N	77.69	4.61

TABLE 5.1 – Résultats sur quelques familles d'instances d'applications

notre intensification est intégrée à MINISAT Eén et Sörenson (2004). Nous observons une croissance du nombre d'instances résolues. Par exemple, si nous considérons la famille $goldb-heqc$, $(MiniSAT+Intensification)$ résoud 2 instances de plus que MINISAT (les instances $goldb-heqc-frg1mul$ et $goldb-heqc-x1mul$). De même sur la famille $9dlx\_vliw\_at$, $(MiniSAT + Intensification)$ résoud 2 instances de plus que MINISAT (les instances $9dlx\_vliw\_at\_b\_iq2$ et $9dlx\_vliw\_at\_b\_iq3$). Globalement, les améliorations sont d'un ordre de magnitude (les instances $velev-vliw-uns-4.0-9C1$, $9dlx\_vliw\_at\_b\_iq1$ et $velev-pipe-sat-1.0-b10$). Ces derniers résultats démontrent clairement que sur certaines familles, MINISAT avec intensification améliore clairement la version de base de MINISAT. La table 5.1 confirme aussi que ces améliorations sont plus significatives sur les instances insatisfiables.

5.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle approche de recherche basée sur l'intensification. Cette recherche basée sur l'intensification est appliquée à chaque redémarrage du solveur SAT. Il collecte les variables rencontrées durant la dernière analyse de conflit en utilisant un ordre prédéfini. Ces variables sont encore selectionnées jusqu'au sommet de l'arbre de recherche pendant le prochain redémarrage. Ce simple principe d'intensification donne des améliorations significatives quand il est intégré aux solveurs SAT de l'état de l'art.

Notre motivation à l'origine de ce travail est de montrer qu'il reste encore des possibilités d'améliorations des heuristiques d'ordonnancement des variables SAT. A notre connaissance, cette direction n'a pas retenue beaucoup d'attentions. Les améliorations obtenues par notre simple stratégie d'intensification suggèrent que des études plus poussées sur le lien entre les redémarrages, les heuristiques d'ordonnancement des variables et l'apprentissage sont nécessaires.

Chapitre 6

Nouvelles Clauses Bi-Assertives et leurs Intégration dans les Solveurs SAT Modernes

Sommaire

DANS CE CHAPITRE, nous proposons une nouvelle approche d'apprentisage de clauses qui est l'une des composantes clés des solveurs SAT modernes. En effet, en traversant le graphe d'implications séparément à partir des littéraux x et $\neg x$, nous dérivons une nouvelle classe de clauses Bi-Assertives qui peuvent conduire à un graphe d'implications plus compact. Ces nouvelles clauses Bi-Assertives sont plus courtes et tendent à induire plus d'implications que les clauses Bi-Assertives classiques. Les résultats expérimentaux montrent que l'exploitation de cette nouvelle classe de clauses Bi-Assertives permet d'améliorer les performances des solveurs issus de l'état de l'art de SAT particulièrement sur les instances crafteds.

6.1 Introduction

Aujourd'hui, le problème SAT a gagné une audience considérable avec l'avénement d'une nouvelle génération de solveurs capablent de résoudre des grandes instances issues du codage des applications du monde réel ainsi que par le fait que ces solveurs constituent d'importants composants de base pour plusieurs domaines,

e.g., SMT (SAT Modulo Théorie), démonstration automatique, comptage de modèles, problème QBF, etc. Ces solveurs communément appelés solveurs SAT modernes CDCL (Conflict Driven, Clause Learning) Moskewicz *et al.* (2001), Eén et Sörenson (2004) sont basées sur la propagation unitaire classique Davis *et al.* (1962) finement combinée avec des structures de données efficaces (ex. Watched literals), des politiques de redémarrages Huang (2007), Gomes *et al.* (1998), Kautz *et al.* (2002), des heuristique de choix de variables basées sur les activités VSIDS (Variable State Independant, Decading Sum) Moskewicz *et al.* (2001), ainsi que l'apprentissage de clauses Marques-Silva et Sakallah (1996b), Moskewicz *et al.* (2001), Zhang *et al.* (2001). L'apprentissage de clauses est actuellement reconnu comme la composante la plus importante des solveurs SAT modernes. Ces solveurs SAT modernes peuvent être vus comme une version étendue de la bien connue procédure DPLL Davis *et al.* (1962) obtenue grâce à ces différentes améliorations. L'algorithme pour la dérivation des clauses à partir des conflits dans ces solveurs SAT est basé sur l'utilisation d'une structure de données importante appelée graphe d'implications Marques-Silva et Sakallah (1996b), Moskewicz *et al.* (2001). Comme évoqué au chapitre 4, Il est important de noter que la bien connue règle de résolution joue encore un rôle important dans l'efficacité des solveurs SAT modernes. Théoriquement, en intégrant l'apprentissage de clauses aux procédures DPLL Davis *et al.* (1962), le solveur SAT obtenu formulé comme un système de preuve s'avère aussi puissant que la résolution générale Pipatsrisawat et Darwiche (2009a; 2010).

Une nouvelle classe de clauses assertives appelée clauses Bi-Assertives qui est une relaxation des clauses assertives a été proposée dans Pipatsrisawat et Darwiche (2008a). Ces clauses Bi-Assertives fréquemment appelées clauses Bi-Assertives classiques permettent de découvrir les implications manquées par les clauses assertives classiques. Elle est définie de la même manière comme une clause assertive. Une clause asssertive contient exactement un littéral du niveau de conflit tandis qu'une clause Bi-Assertive contient exactement deux littéraux du niveau de conflit. Pour plus d'amples détails autours des clauses Bi-Assertives classiques, nous référons les lecteurs à Pipatsrisawat et Darwiche (2008a).

L'approche proposée dans ce chapitre est basée sur une analyse de conflit séparée. En effet, en traversant le graphe d'implications séparément à partir de x et $\neg x$, nous dérivons une nouvelle classe de clauses Bi-Assertives. Ces clauses diffèrent de celles proposées par Pipatsrisawat et Darwiche Pipatsrisawat et Darwiche (2008a) car elles ne sont pas des clauses falsifiées c'est à dire satisfaites par l'interprétation partielle courante. Ces clauses ne peuvent être dérivées par le parcours traditionnel du graphe d'implications. De plus, nos clauses Bi-Assertives peuvent être utilisées pour obtenir un graphe d'implications plus compact. Elles sont plus courtes et tendent à induire plus d'implications que les clauses Bi-Assertives classiques.

Ce chapitre est organisé comme suit. Afin de permettre une bonne compréhension de notre approche d'apprentissage, nous présentons tout d'abord quelques notions liées à l'apprentissage par analyse de conflit basée sur l'analyse du graphe d'implications et aux clauses Bi-Assertives classiques. Ensuite, notre approche est présentée dans la section 6.3. Finalement avant de conclure, les résultats expérimentaux obtenus sur les instances des récentes compétitions SAT démontrant la faisabilité de cette approche sont présentés.

Les travaux présentés dans ce chapitre ont donné lieu à deux publications Jabbour *et al.* (2013; 2014a).

6.2 Clauses Bi-Assertives classiques

Les clauses Bi-Assertives classiques sont dérivées au cours de l'analyse de conflit en utilisant le graphe d'implications. Avant d'introduire formellement leur définition, il est important de revenir plus en détails sur les notions d'analyse de conflit et de graphe d'implications et plus particulièrement sur le processus de dérivation de la clause assertive classique afin de mieux situer le contexte de ce travail.

Comme nous l'avons vu dans les sections précédentes, le graphe d'implications est une représentation standard convenablement utilisée pour analyser les conflits dans les solveurs SAT modernes. Dans la suite, nous rappelons les définitions formelles du graphe d'implications, des clauses assertives classiques et comment ces clauses sont dérivées Audemard *et al.* (2008a).

Définition 6.1 (graphe d'implications)**.** *Soient $\mathcal{F}$ une formule* CNF*, $\mathcal{I}$ une interprétation partielle. Soit exp l'ensemble des explications pour des littéraux déduits par propagation unitaire $\mathcal{I}$. Le graphe d'implications associé à $\mathcal{F}$, $\mathcal{I}$ et exp est $\mathcal{G}_{\mathcal{F}}^{\mathcal{I},exp} = (\mathcal{N}, \mathcal{A})$ tel que :*

- $\mathcal{N} = \{x \in \mathcal{I}\}$*, c'est-à-dire un nœud pour chaque littéral de $\mathcal{I}$, de décision ou propagé ;*
- $\mathcal{A} = \{(x, y) \mid x \in \mathcal{I}, y \in \mathcal{I}, x \in exp(y)\}$.

Dans la suite de ce chapitre, pour des raisons de simplicité, un graphe d'implications sera simplement noté $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}}$. nous noterons aussi m le niveau de conflit. Pour une meilleure comprehension, nous illustrons à nouveau le graphe d'implications à travers l'exemple suivant :

Exemple 6.1. *$\mathcal{G}_{\mathcal{F}}^{\mathcal{I}}$, représenté par la figure 6.1 est un graphe d'implications pour la*

formule $\mathcal{F}$ et l'interprétation partielle $\mathcal{I}$ données ci-dessous : $\mathcal{F} \supseteq \{c_1, \ldots, c_{12}\}$

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

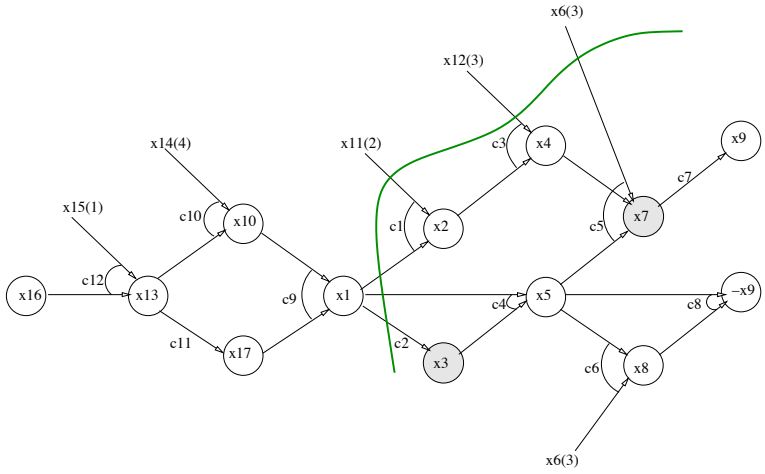


FIGURE 6.1 – graphe d'implications $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}} = (\mathcal{N}, \mathcal{A})$

$\mathcal{I} = \{\langle \ldots x_{15}^1 \ldots \rangle \langle (x_{11}^2) \ldots \ldots \rangle \langle (x_{12}^3) \ldots x_6^3 \ldots \rangle$
$\langle (x_{14}^4), \ldots \rangle \langle (x_{16}^5), x_{13}^5, x_{10}^5, x_{17}^5, x_1^5 \ldots \rangle\}$.

Un conflit est rencontré sur les littéraux x_9 et $\neg x_9$, le niveau de conflit est 5 et $\mathcal{I}(\mathcal{F}) = faux$.

Définition 6.2 (clause assertive)**.** *Une clause c de la forme $(\alpha \vee x)$ est appelée clause assertive si et seulement si $\mathcal{I}(c) = \bot$, $niv(x) = m$ et $\forall y \in \alpha, niv(y) < niv(x)$. x est appelé littéral* assertif, *tandis que $niveauAssertion(c) = max\{level(\neg y) \mid y \in \alpha\}$ est le niveau d'assertion du littéral x.*

L'analyse de conflit dont nous rappelons également la définition ici est le résultat de l'application de la résolution à partir de la clause conflit en utilisant différentes implications encodées dans le graphe d'implications. On appelle ce processus dérivation de la clause assertive (DCA en court).

Définition 6.3 (dérivation de la clause assertive)**.** *Une dérivation de la clause assertive π est une sequence de clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfaisant les conditions suivantes :*

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

Considérons à nouveau l'exemple 6.1. Le parcours du graphe d'implications $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathcal{N}, \mathcal{A})$ (voir Figure 6.1) conduit à la dérivation de la clause assertive $\langle \sigma_1, \ldots \sigma_7 \rangle$ où :

- $\sigma_1 = \eta[x_9, c_7, c_8] = (\neg {x_5}^5 \vee \neg {x_7}^5 \vee \neg {x_8}^5)$
- $\sigma_2 = \eta[x_8, \sigma_1, c_6] = (\neg x_6^3 \vee \neg {x_5}^5 \vee \neg {x_7}^5)$
- $\sigma_3 = \eta[x_7, \sigma_2, c_5] = (\neg x_6^3 \vee \neg {x_5}^5 \vee \neg {x_4}^5)$
- $\sigma_4 = \eta[x_4, \sigma_3, c_3] = (\neg x_{12}^3 \vee \neg x_6^3 \vee \neg {x_5}^5 \vee \neg {x_2}^5)$
- $\sigma_5 = \eta[x_5, \sigma_4, c_4] = (\neg x_{12}^3 \vee \neg x_6^3 \vee \neg {x_2}^5 \vee \neg {x_3}^5 \vee \neg x_1^5)$
- $\sigma_6 = \eta[x_2, \sigma_5, c_1] = (\neg x_{11}^2 \vee \neg x_{12}^3 \vee \neg x_6^3 \vee \neg {x_3}^5 \vee \neg x_1^5)$
- $\sigma_7 = \eta[x_3, \sigma_6, c_2] = (\neg {x_{11}}^2 \vee \neg {x_{12}}^3 \vee \neg {x_6}^3 \vee \neg {x_1}^5)$

La clause σ_7 est la prémière clause rencontrée qui contient un seul littéral $\neg x_1$ du niveau de conflit courant 5. Notons que cette résolvante est fausse sous l'interprétation partielle $\mathcal{I}$. Par conséquent, le littéral $\neg x_1$ est impliqué au niveau 3 qui correspond au niveau maximum de tous les autres littéraux (${\neg x_{11}}^2$, ${\neg x_{12}}^3$ et ${\neg x_6}^3$) de σ_7. Les solveurs SAT ajoutent la clause σ_7 à la base des clauses apprises, effectuent un retour-arrière non chronologique au niveau 3 et affectent le littéral assertif $\neg x_1$ à la valeur de vérité vraie par propagation unitaire. Le nœud x_1 correspondant au littéral assertif $\neg x_1$ est appelé Premier Unique Point d'Implication (First UIP).

Nous introduisons à présent une propriété importante appelée *1−empowerment* Pipatsrisawat et Darwiche (2008a), qui caractérise les clauses assertives classiques.

Définition 6.4 (1−Empowerment)**.** *Soit* c *une clause de la forme* $(\alpha \vee l)$, *telle que* l *est un littéral et* α *une disjonction de littéraux (une sous-clause). La clause* c *est* 1−empowerment *par rapport à la formule* CNF $\mathcal{F}$ *via le littéral* l *si et seulement si :*

1. $\mathcal{F} \models (\alpha \vee l)$ *: La clause* c *est impliquée par* $\mathcal{F}$
2. $\mathcal{F} \wedge \neg\alpha \nvdash l$ *: Le littéral* l *ne peut pas être dérivé de la formule* $\mathcal{F} \wedge \neg\alpha$ *en utilisant la propagation unitaire.*

Toute clause assertive $c = (\alpha \vee l)$ satisfait la propriété *1−empowerment*. En effet, comme c est dérivée de la formule $\mathcal{F}$ par résolution, alors c est une conséquence de $\mathcal{F}$ (condition 1 de la définition 6.4). La seconde condition de la définition

6.4 est aussi satisfaite comme le littéral l n'est pas dérivé au niveau d'assertion où la sous-clause α est falsifiée (condition 2 de la définition 6.4). Cette propriété exprime que le littéral l ne peut pas être déduit par propagation unitaire de $\mathcal{F}$ sans l'ajout de la clause c à $\mathcal{F}$. Cependant, si on ajoute la clause assertive à la base des clauses apprises, on déduit par propagation unitaire que le littéral l doit être assigné à vrai au niveau d'assertion. Un autre type de clauses assertives appelées clauses Bi-Assertives fut introduit dans Pipatsrisawat et Darwiche (2008a).

Définition 6.5 (Clauses Bi-Assertives)**.** *Une clause c de la forme $(\alpha \vee x \vee y)$ est appelée clause Bi-Assertive si $\mathcal{I}(c) =$ `faux` , $level(\alpha) < m$ et $level(y) = level(x) = m$.*

Il est important de noter qu'une clause Bi-Assertive classique est fausse sous l'interprétation courante $\mathcal{I}$. Pour dériver de telles clauses Bi-Assertives à partir d'un conflit, on suit exactement la même dérivation par résolution. La seule différence est que le processus se termine quand la résolvante courante est une clause Bi-Assertive. Comme toute clause Bi-Assertive ne contribue pas à la progapation unitaire, dans Pipatsrisawat et Darwiche (2008a), les auteurs proposent d'apprendre la prémière clause Bi-Assertive qui satisfait la propriété *1−empowerment* respectivement avec les clauses utilisées pour sa dérivation.

Exemple 6.2. *Considérons par exemple la formule* CNF $\mathcal{F} = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$ *et la clause $(x_2 \vee x_4)$ qui est impliquée par la formule $\mathcal{F}$. L'ajout du littéral $\neg x_4$ à $\mathcal{F}$ ne permet pas à la propagation unitaire de dériver x_2 bien que x_2 soit impliqué par $\mathcal{F} \wedge \neg x_4$. Cette dérivation devient possible si nous ajoutons la clause $(x_2 \vee x_4)$ à $\mathcal{F}$. Ainsi la clause $(x_2 \vee x_4)$ est* 1−empowering *respectivement avec $\mathcal{F}$ via le littéral x_2. Cependant la clause $(x_2 \vee x_3)$ n'est pas* 1−empowering *respectivement avec la formule $\mathcal{F}$, car $\mathcal{F} \wedge \neg x_2 \vdash x_3$ et $\mathcal{F} \wedge \neg x_3 \vdash x_2$.*

Dans la section suivante, nous proposons une nouvelle classe de clauses Bi-Assertives plus courtes et qui tendent à induire plus d'implications que les clauses Bi-Assertives classiques. Ces nouvelles clauses Bi-Assertives conduisent à un graphe d'implications plus compact.

6.3 Une nouvelle classe de clauses Bi-Assertives

Dans cette section, nous montrons premièrement comment dériver les nouvelles clauses Bi-Assertives. Ensuite, nous démontrons que leur utilisation peut conduire à obtenir plus d'implications qu'avec les clauses Bi-Assertives classiques. Illustrons cette nouvelle approche proposée en utilisant un simple exemple.

6.3.1 Motivation

Soit $\mathcal{F}$ une formule CNF, $\mathcal{I}$ une affectation partielle. Supposons que nous avons un graphe d'implications $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}}$ associé à $\mathcal{F}$ et $\mathcal{I}$. Pour des raisons de simplicité, la figure 6.2 représente le graphe d'implications restreint au sous-graphe induit par les nœuds entre le conflit et le Premier Unique Point d'Implication (First UIP).
Supposons que le niveau de conflit est 5.

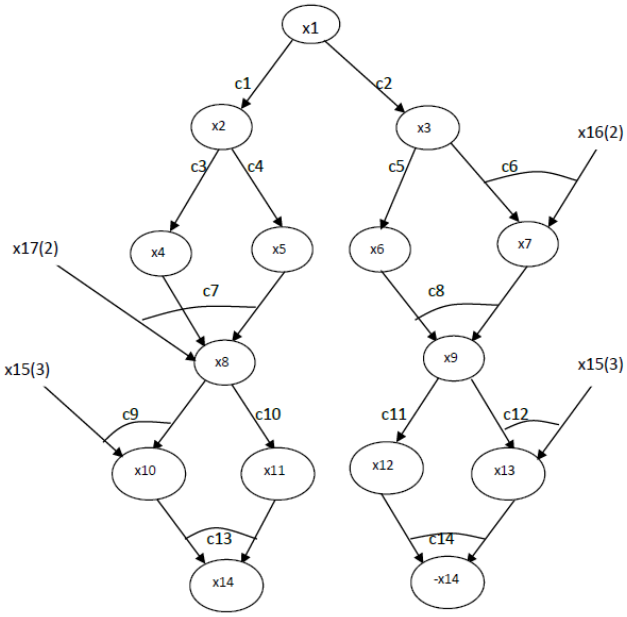


FIGURE 6.2 – sous-graphe d'implications $\mathcal{G}_{\mathcal{F}}^{\mathcal{I}} = (\mathcal{N}, \mathcal{A})$

En traversant le graphe d'implications séparément à partir des littéraux x_{14} et $\neg x_{14}$ jusqu'au Premier Unique Point d'Implication x_1, nous dérivons de nouvelles clauses Bi-Assertives δ_1, δ_2, δ_3 et δ_4 comme suit :

- $\sigma_1 = \eta[x_{10}, c_{13}, c_9] = (\neg x_{15}^3 \vee \neg x_{11}^5 \vee \neg x_8^5 \vee x_{14}^5)$
- $\delta_1 = \eta[x_{11}, \sigma_1, c_{10}] = (\neg x_{15}^3 \vee \neg x_8^5 \vee x_{14}^5)$
- $\sigma_2 = \eta[x_4, c_7, c_3] = (\neg x_{17}^2 \vee \neg x_5^5 \vee \neg x_2^5 \vee x_8^5)$
- $\delta_2 = \eta[x_5, \sigma_2, c_4] = (\neg x_{17}^2 \vee \neg x_2^5 \vee x_8^5)$
- $\sigma_3 = \eta[x_{12}, c_{14}, c_{11}] = (\neg x_{13}^5 \vee \neg x_9^5 \vee \neg x_{14}^5)$
- $\delta_3 = \eta[x_{13}, \sigma_3, c_{12}] = (\neg x_{15}^3 \vee \neg x_9^5 \vee \neg x_{14}^5)$
- $\sigma_4 = \eta[x_6, c_8, c_5] = (\neg x_3^5 \vee \neg x_7^5 \vee x_9^5)$

– $\delta_4 = \eta[x_7, \sigma_4, c_6] = (\neg x^2_{16} \vee \neg x^5_3 \vee x^5_9)$

Les nouvelles clauses Bi-Assertives δ_1, δ_2, δ_3, δ_4 forment un ensemble de clauses connexes. Plus précisement, la clause δ_2 est liée à la clause δ_1 par la variable x_8, δ_1 est liée à δ_3 par la variable x_{14} et δ_3 est liée à δ_4 par la variable x_9. Comme nous pouvons l'observer, l'ensemble des clauses Bi-Assertives forment une chaîne de clauses connectées.

Il est important de noter que les clauses Bi-Assertives classiques qui peuvent être obtenues du graphe d'implications de la figure 6.2 sont les suivantes :

– $\delta'_1 = (\neg x^3_{15} \vee \neg x^5_8 \vee \neg x^5_9)$

– $\delta'_2 = (\neg x^3_{15} \vee \neg x^2_{17} \vee \neg x^5_2 \vee \neg x^5_9)$

– $\delta'_3 = (\neg x^3_{15} \vee \neg x^2_{16} \vee \neg x^5_8 \vee \neg x^5_3)$

– $\delta'_4 = (\neg x^3_{15} \vee \neg x^2_{17} \vee \neg x^2_{16} \vee \neg x^5_2 \vee \neg x^5_3)$.

Notons qu'en utilisant le schéma d'apprentissage classique de clauses (schéma First UIP), on peut dériver la clause assertive $(\neg x^2_{17} \vee \neg x^2_{16} \vee \neg x^3_{15} \vee \neg x^5_1)$ qui fournit un niveau de backjumping égal à 3.

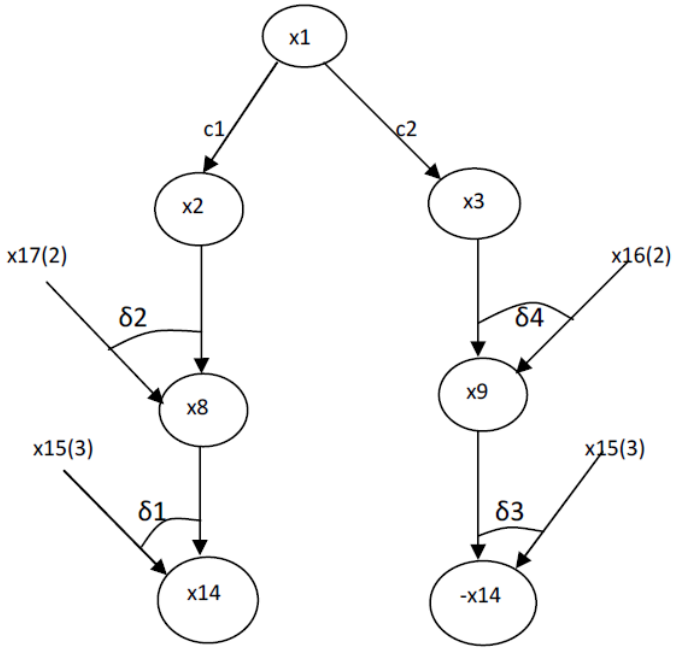


FIGURE 6.3 – Nouveau graphe d'implications plus compact obtenu de $\mathcal{G}^{\mathcal{I}}_{\mathcal{F}} = (\mathcal{N}, \mathcal{A})$ en utilisant les nouvelles clauses Bi-Assertives.

A travers la figure 6.3, nous montrons que le graphe d'implications original (figure 6.2) peut être reécrite de façon plus compact.

Considérons à présent que les nouvelles clauses Bi-Assertives δ_1, δ_2, δ_3 et δ_4 sont ajoutées à la base des clauses apprises et supposons qu'au niveau 3 le littéral x_2 est assigné à vrai. Il est facile de voir que tous les littéraux x_8, x_{14}, $\neg x_9$, $\neg x_3$ et $\neg x_1$ (voir figure 6.3) ensemble avec les littéraux x_4, x_5, x_{10}, x_{11} seront impliqués par propagation unitaire. Ces derniers littéraux propagés sont impliqués grâce aux clauses initiales de la formule qui sont c_3, c_4, c_9 et c_{10}.
Cependant, si nous considérons le graphe d'implications original (figure 6.2), nous dérivons par propagation unitaire uniquement les littéraux x_8, x_{14}, x_4, x_5, x_{10}, x_{11} et pas les littéraux $\neg x_9$, $\neg x_3$ et $\neg x_1$.

Supposons maintenant que toutes les clauses Bi-Assertives classiques δ'_1, δ'_2, δ'_3 et δ'_4 sont ajoutées à la base des clauses apprises. L'affectation de x_2 à vrai au niveau 3 conduit au même ensemble d'implications par propagation unitaire. Cependant, si nous décidons d'assigner x_{14} à vrai au niveau 3, en utilisant les nouvelles clauses Bi-Assertives, nous dérivons les littéraux $\neg x_9$, $\neg x_3$ et $\neg x_1$ tandis qu'avec les clauses Bi-Assertives classiques aucun littéral n'est impliqué par propagation unitaire.

Autre différence importante qui peut être faite ici est que la dérivation de toutes les clauses Bi-Assertives classiques possibles est quadratique dans le pire des cas alors que les nouvelles clauses Bi-Assertives peuvent être dérivées en temps linéaire (voir figure 6.3). Par conséquent, la recherche de toutes les clauses Bi-Assertives classiques prend plus de temps que la recherche des nouvelles clauses Bi-Assertives proposées.

En effet, en utilisant la méthode d'apprentissage classique, nous ne pouvons pas dériver à la fois les clauses δ'_2 et δ'_3. Plus précisement, pour dériver δ'_2 (respectivement δ'_3) , nous devons passer par une étape de résolution sur le littéral $\neg x_8$ (respectivement $\neg x_9$) et la clause δ'_1 (respectivement δ'_1) . Ainsi, la résolvante courante obtenue ne contiendra plus $\neg x_8$ (respectivement $\neg x_9$), il ne sera donc plus possible dans la suite du processus d'apprentissage d'obtenir la clause δ'_3 (respectivement δ'_2) qui contient le littéral $\neg x_9$ (respectivement $\neg x_8$). De plus, nos nouvelles clauses Bi-Assertives sont plus courtes que celles classiques.

A partir de cet exemple illustratif, on voit que l'ajout des nouvelles clauses Bi-Assertives à la base des clauses apprises permet de dériver plus d'implications que l'ajout des clauses Bi-Assertives classiques. Nous pouvons également observer que les clauses Bi-Assertives proposées établissent un lien entre le littéral assertif $\neg x_1$ et les deux littéraux conflits x_{14} et $\neg x_{14}$.

6.3.2 Analyse de conflit séparée : formulation générale

On donne à présent une présentation formelle de notre approche de dérivation des nouvelles clauses Bi-Assertives. Dans toutes les définitions suivantes, nous

considérons m comme le niveau de conflit courant.

Dans notre approche, à partir d'un simple conflit, nous dérivons plusieurs nouvelles clauses Bi-Assertives en traversant le graphe d'implications séparement à partir des deux littéraux conflits x et $\neg x$ jusqu'au Premier Unique Point d'Implications (First UIP). Ces nouvelles clauses sont ensuite ajoutées à la base des clauses apprises pour intensifier la propagation unitaire.

Notons que nous avons seulement besoin d'un simple parcours du graphe d'implications pour dériver à la fois la clause assertive classique et l'ensemble des nouvelles clauses Bi-Assertives. En effet, la clause assertive classique peut être progressivement générée des nouvelles clauses Bi-Assertives. De cette manière, nous n'avons aucun surcoût de recherche additionnel dû à la dérivation de ces nouvelles clauses Bi-Assertives.

Définition 6.6 (dérivation de la clause Bi-Assertive)**.** *Soit x un littéral conflit, nous définissons la dérivation de la clause Bi-Assertive à partir de x comme la séquence de clauses $C_x = \langle \sigma_1^x, \sigma_2^x, \ldots, \sigma_k^x \rangle$ satisfaisant les conditions suivantes :*

1. *σ_1^x est dérivée par résolution sur z entre $\overrightarrow{raison}(x)$ et $\overrightarrow{raison}(z)$ où $\neg z \in \overrightarrow{raison}(x)$;*
2. *σ_i^x pour tout $i \in 2..k$, est construite par résolution sur un littéral $z \in \sigma_{i-1}^x$ tel que $niv(z) = m$ et pour lequel $\overrightarrow{raison}(\overline{z})$ est définit.*
   La clause σ_i^x est définie comme $\eta[z, \sigma_{i-1}^x, \overrightarrow{raison}(\overline{z})]$;
3. *σ_k^x est une clause Bi-Assertive.*

Définition 6.7 (littéral Bi-Assertif)**.** *Soit x un littéral conflit, soit $C_x = \langle \sigma_1^x, \sigma_2^x, \ldots, \sigma_k^x \rangle$ une dérivation de la clause Bi-Assertive, nous définissons un littéral Bi-Assertif comme un littéral $b \in \sigma_k^x$ tel que $b \neq x$ et $niv(b) = m$.*

Si nous considérons à nouveau le graphe d'implications de la figure 6.2, l'ensemble des différents littéraux Bi-Assertifs sera $\{x_8, x_9, x_2, x_3, x_1\}$.

Propriété 6.1. *Soientt $\mathcal{G}$ un graphe d'implications et x un littéral conflit de $\mathcal{G}$. Soient B^x et $B^{\neg x}$ les deux ensembles de clauses Bi-Assertives générées à partir des deux littéraux conflits x et $\neg x$ respectivement. Au niveau du backjumping, si un littéral Bi-Assertif y de B^x (respectivement $B^{\neg x}$) est assigné à vrai, alors tous les littéraux $\neg z$ tels que z est un littéral Bi-Assertif de $B^{\neg x}$ (respectivement B^x) seront impliqués.*

Preuve 6.1. *La preuve est triviale. Une illustration de la preuve est donnée dans l'exemple de motivation précédent.*

Il est facile de voir que le nombre de nouvelles clauses Bi-Assertives qui peuvent être générées de tout graphe d'implications est au moins égal à 2. En effet, si nous considérons un graphe d'implications $\mathcal{G}$ avec un littéral conflit x et y comme littéral assertif (schéma First UIP), il est facile de voir qu'on peut dériver au moins deux clauses Bi-Assertives $b_1 = (\alpha \vee y \vee x)$ et $b_2 = (\beta \vee y \vee \neg x)$. Dans ce cas la clause assertive résultante est $a = (\alpha \vee \beta \vee y)$.

Propriété 6.2. *Soit $\mathcal{G}$ un graphe d'implications, x un littéral conflit de $\mathcal{G}$. Soient N^x et $N^{\neg x}$ les deux ensembles de clauses encodées dans $\mathcal{G}$ entre le First UIP et les deux littéraux conflits x et $\neg x$ respectivement. La complexité temporelle nécessaire pour générer toutes nos nouvelles clauses Bi-Assertives est en $\mathcal{O}(|N^x| + |N^{\neg x}|)$ dans le pire des cas.*

Preuve 6.2. *Comme nos nouvelles clauses Bi-Assertives sont générées en remontant en parallèle les deux branches (branche contenant le littéral x et celle contenant $\neg x$) du graphe d'implications $\mathcal{G}$ à partir des deux littéraux conflits x et $\neg x$, chaque clause de N^x et $N^{\neg x}$ est impliquée dans une étape de résolution. Ainsi la complexité temporelle nécessaire pour la dérivation de toutes les nouvelles clauses Bi-Assertives est en $\mathcal{O}(|N^x| + |N^{\neg x}|)$ dans le pire des cas.*

Si nous considérons les clauses Bi-Assertives classiques Pipatsrisawat et Darwiche (2008a), et souhaitons générer toutes ces clauses, la complexité temporelle nécessaire pour éffectuer cette opération sera quadratique dans le pire des cas.

Propriété 6.3. *Soit $\mathcal{G}$ un graphe d'implications, x un littéral conflit de $\mathcal{G}$. Soient N^x et $N^{\neg x}$ les deux ensembles de clauses encodées dans $\mathcal{G}$ entre le First UIP et les deux littéraux conflits x et $\neg x$ respectivement. La complexité temporelle nécessaire pour générer toutes les clauses Bi-Assertives classiques est en $\mathcal{O}(|N^x| \times |N^{\neg x}|)$ dans le pire des cas.*

Preuve 6.3. *Le pire des cas est rencontré lorsque que nous avons un graphe d'implications $\mathcal{G}$ avec $|N^x|$ et $|N^{\neg x}|$ littéraux Bi-Assertifs. La forme d'un tel graphe est représentée par la figure 6.3. Cette figure correspond à un graphe d'implications dans lequel tous les nœuds sont des littéraux Bi-Assertifs. Comme le processus de génération des clauses Bi-Assertives classiques commence par effectuer une première étape de résolution entre les deux clauses impliquant les littéraux x et $\neg x$, pour générer toutes les clauses Bi-Assertives classiques, nous gardons alternativement le littéral Bi-Assertif de la partie du graphe liée à x (respectivement à $\neg x$), et nous traversons toute l'autre partie du graphe liée à $\neg x$ (respectivement à x). Par conséquent, la complexité temporelle nécessaire pour la dérivation de toutes les clauses Bi-Assertives* ***classiques*** *est en $\mathcal{O}(|N^x| \times |N^{\neg x}|)$.*

Il est important de noter que les clauses Bi-Assertives classiques et les nouvelles clauses Bi-Assertives proposées dans ce chapitre diffèrent sur certains aspects. Premièrement, les nouvelles clauses Bi-Assertives (respectivement clauses Bi-Assertives classiques) sont satisfaites (respectivement falsifiées) sous l'interprétation courante. Deuxièmement, les deux classes de clauses sont générées en utilisant des traversées différentes du graphe d'implications. Par conséquent, les ensembles de clauses Bi-Assertives générées par les deux approches sont différents (Voir l'exemple de motivation donné à la section 6.3.1).

6.3.3 Exploitation des nouvelles clauses Bi-Assertives

Au cours de la recherche, quand un conflit est rencontré sur un littéral x, nous l'analysons en traversant le graphe d'implications séparément à partir de x et $\neg x$. Toutes les clauses Bi-Assertives dérivées au cours de cette analyse sont stockées dans une nouvelle base de clauses Bi-Assertives apprises. Ces nouvelles clauses sont gérées de la même manière que les clauses de la base des clauses apprises classiques. Nous exploitons une stratégie de réduction de la base des clauses apprises identique à celle implémentée dans MINISAT 2.2 Eén et Sörenson (2004) qui est le solveur le plus utilisé actuellement. En effet, dans ce solveur, la fonction de réduction de la base des clauses apprises est appelée une fois que la taille de la base dépasse un certain seuil. La clause assertive classique est aussi ajoutée à la base des clauses apprises classiques.

6.4 Expérimentations

Les expérimentations ont été réalisées sur un large panel d'instances industrielles et crafteds issues des dernières compétitions SAT. Toutes les instances sont simplifiées par pré-traitement `SatElite` Eén et Biere (2005). Afin d'étudier l'impact de notre approche, nous l'avons implémentée dans le solveur MINISAT 2.2 Eén et Sörenson (2004) et nous avons effectué une comparaison entre le solveur original et celui amélioré avec l'apprentissage des nouvelles clauses Bi-Assertives que nous appelons `Minisat+NB`. Tous les tests ont été effectués sur un cluster Xeon 3.2GHz (32 GB RAM). Les résultats du temps de calcul sont indiqués en secondes. Pour ces expérimentations, le temps de calcul limite a été fixé à 4 heures.

6.4.1 Problèmes crafteds

Pour ces problèmes, nous utilisons un ensemble d'instances crafteds issues de la compétition SAT 2011. Le schéma (en échelle logarithmique) donné dans la par-

tie haute de la figure 6.4 détaille les résultats de `Minisat` et `Minisat+NB` sur chaque instance crafted. L'axe x (respectivement y) correspond au temps CPU tx (respectivement ty) obtenu par `Minisat` (respectivement `Minisat+NB`).

Chaque point de coordonnées (tx, ty) correspond à une instance SAT. Les points en-dessous (respectivement au-dessus) la diagonale indique les instances résolues plus rapidement en utilisant notre approche c'est à dire $ty < tx$ (respectivement $ty > tx$). La majorité des points sur la figure se trouvent en-dessous la diagonale, ce qui signifie que `Minisat+NB` est meilleur. Cette figure montre clairement le gain de temps obtenu grâce à notre approche. Globalement, sur les instances de type crafted, notre approche apporte des améliorations intéressantes.

La partie bas de la figure 6.4 montre les mêmes résultats avec une représentation différente donnant pour chaque technique le nombre d'instances résolues (# instances) en moins de t secondes. Cette figure confirme l'efficacité de notre approche d'apprentissage sur ces problèmes. De cette figure, nous pouvons observer que `Minisat+NB` est généralement plus rapide et résoud 15 instances de plus que `Minisat`.

Une analyse plus fine de la partie haute de la figure 6.4 montre que `Minisat+NB` résoud plus rapidement 81 instances que `Minisat`, qui lui même résoud 51 instances plus efficacement que son opposé.

6.4.2 Problèmes industriels

Pour ces problèmes, nous utilisons un ensemble d'instances industrielles issues de la SAT Challenge 2012. Le schéma (en échelle logarithmique) représenté par la figure 6.5 détaille les résultats de `Minisat` et `Minisat+NB` sur chaque instance industrielle. Sur ces instances industrielles, les résultats montrent que les deux solveurs ont un comportement similaire. Ceci confirme que lorsque le graphe d'implications ne contient pas de littéraux Bi-Assertifs intermédiaires sur les deux côtés des littéraux conflit, notre approche ne cause pas de surcoût additionnel. En effet, le graphe d'implications est traversé seulement une fois. Globalement, nous pouvons voir que l'addition de notre processus d'apprentissage à `Minisat` améliore ses performances sur certaines familles d'instances.

La table 6.1 représente un focus sur quelques familles industrielles. Pour ces familles, les gains sont relativement importants. Par exemple, si nous conséderons la famille $vmpc$, nous pouvons voir que notre approche d'apprentissage permet d'avoir un gain d'un ordre de magnitude avec `Minisat+NB` (instances 31, and 33). Sur la même famille, `Minisat+NB` résoud une instance de plus que `Minisat`. Sur la famille $manol$, nous pouvons également voir que `Minisat+NB` résoud 6 instances de plus que `Minisat`. Sur la famille $velev$, `Minisat+DS` est meilleur sur les 4

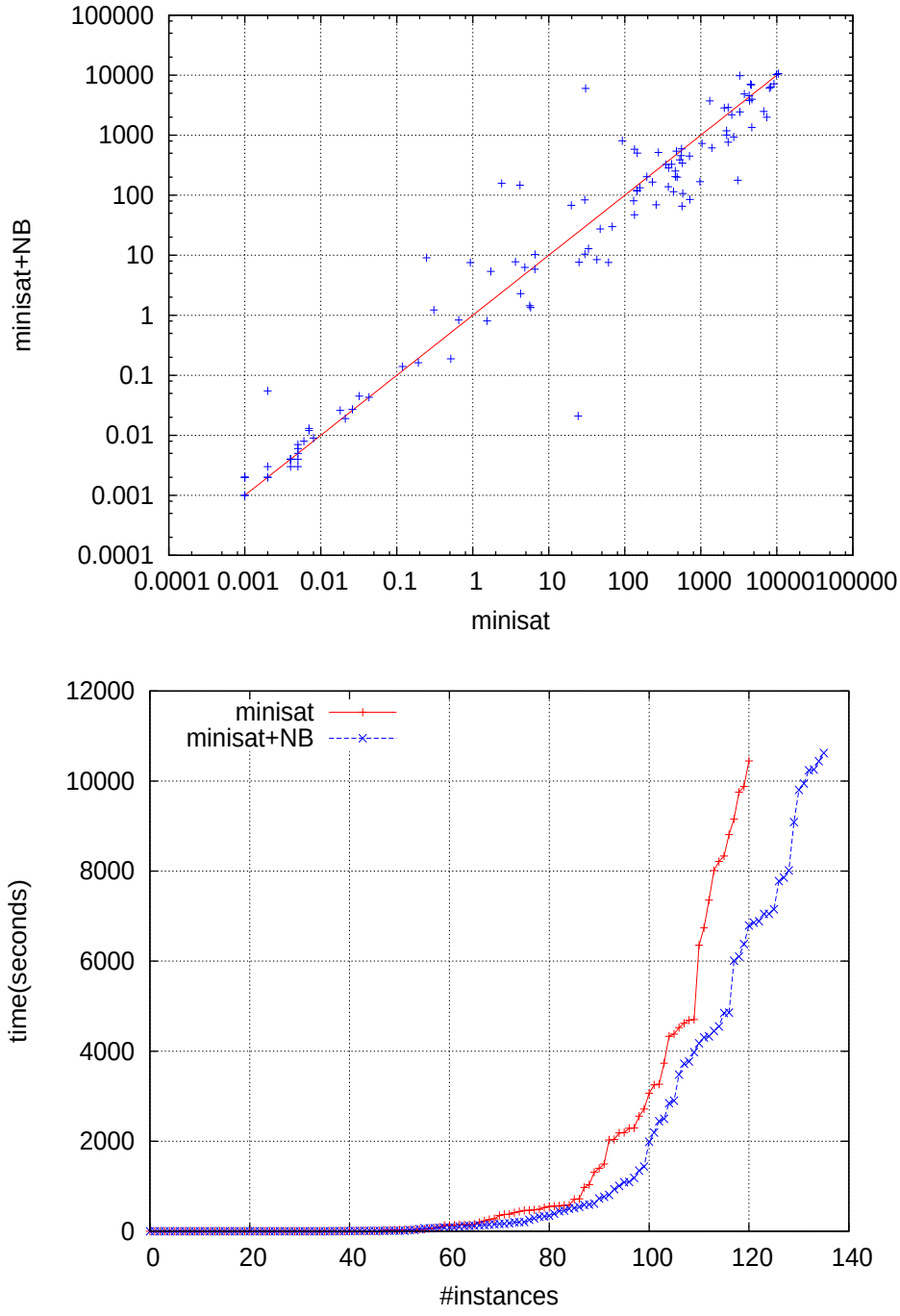


FIGURE 6.4 – **Problèmes Crafted : Minisat vs Minisat+NB**

instances résolues par `Minisat+NB` et `Minisat` et résoud 3 instances de plus que `Minisat`. Globalement, on peut voir que sur certaines familles, `Minisat+NB` est plus rapide et résoud plus de problèmes que `Minisat`.

6.5 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle approche pour l'apprentissage des clauses. Plus précisement, à chaque conflit, nous traversons le graphe d'implications séparément à partir de x et $\neg x$ (x le littéral conflit) jusqu'au Premier Unique Point d'Implication (First UIP). Au cours de cette traversée, nous dérivons une nouvelle classe de clauses Bi-Assertives qui peuvent conduire à un graphe d'implications plus compact. Plus intéressant, ces nouvelles clauses Bi-Assertives tendent à être plus courtes et induisent plus d'implications que les clauses Bi-Assertives classiques Pipatsrisawat et Darwiche (2008a).

Les résultats expérimentaux montrent que l'exploitation de cette nouvelle classe de clauses Bi-Assertives permet d'améliorer les performances des solveurs issus de l'état de l'art de SAT. Notre approche bénéficie particulièrement aux instances crafted et permet des améliorations significatives sur certaines familles d'instances industrielles.

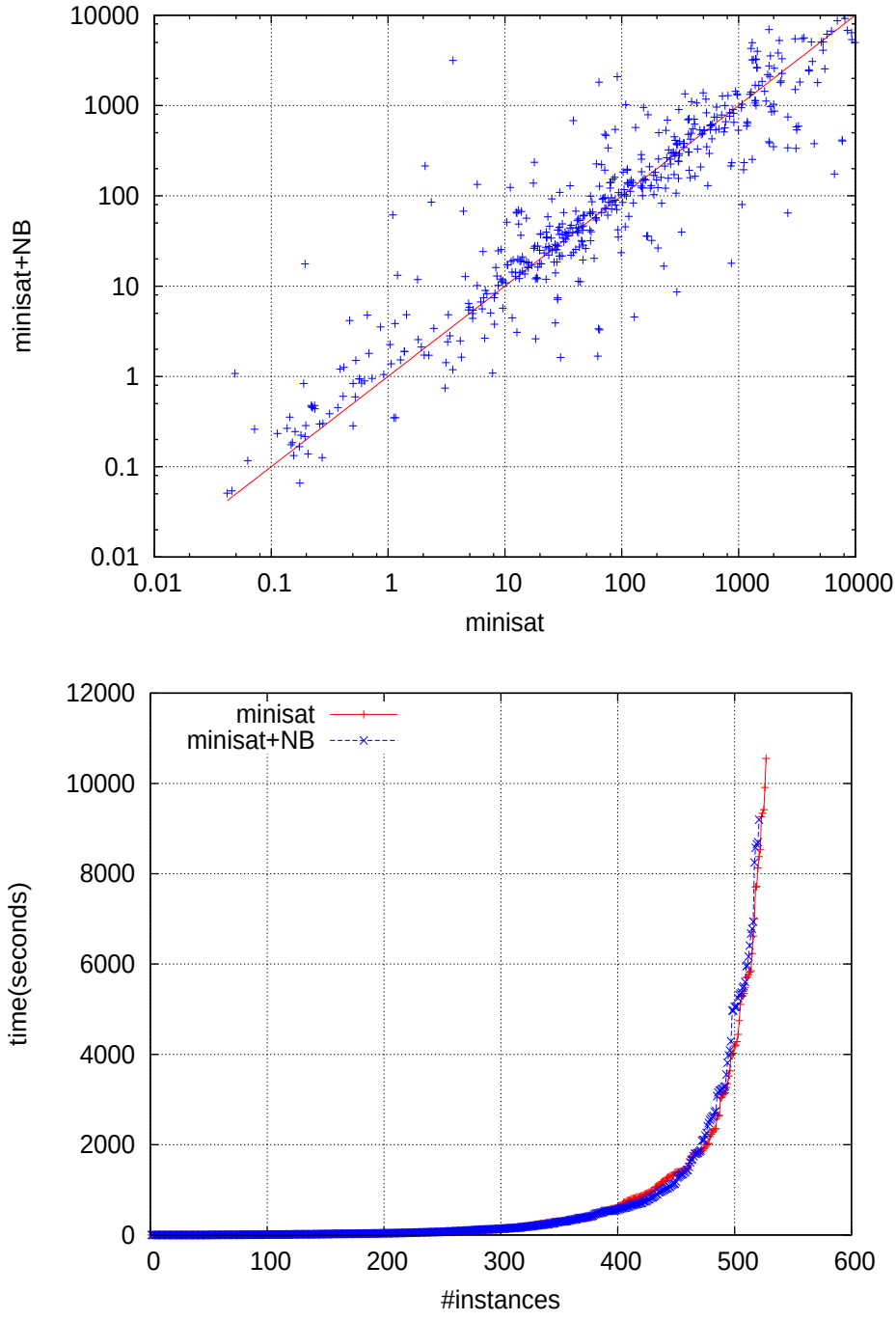


FIGURE 6.5 – **Problèmes Industriels : Minisat vs Minisat+NB**

families	**Minisat**	**Minisat+NB**
vmpc_26	**14.14**	21.06
vmpc_27	42.62	**11.31**
vmpc_28	98.99	**23.45**
vmpc_29	**73.41**	465.88
vmpc_30	**467.25**	535.62
vmpc_31	7707.81	**415.29**
vmpc_32	**1822.31**	2742.98
vmpc_33	6617.32	**174.37**
vmpc_34	-	**120.65**
manol-pipe-f7idw	4442.4	**378.08**
manol-f9b	**125.13**	139.83
manol-f9bidw	-	**670.52**
manol-f9nidw	-	**680.25**
manol-f8nidw	3105.14	**335.41**
manol-f8idw	-	**1606.05**
manol-f8bidw	-	**181.76**
manol-fn	**36.11**	44.09
manol-f10ni	**392.21**	492.48
manol-f10bi	491.96	**490.90**
manol-f10n	**291.57**	296.35
manol-f10nidw	-	**2104.96**
manol-f10i	376.61	**305.58**
manol-f10bid	1381.6	**1066.62**
manol-f10id	**1279.94**	1402.43
manol-f10bidw	-	**1742.06**
velev-7pipe_q0_k	4747.46	**1796.33**
velev-11pipe_q0_k	-	-
velev-14pipe_q0_k	-	-
velev-13pipe_q0_k	-	-
velev-10pipe_q0_k	-	**5342.33**
velev-9pipe_q0_k	545.58	**129.849**
velev-15pipe_q0_k	-	-
velev-8pipe_q0_k	-	**5316.14**
velev-12pipe_k	-	-
velev-8pipe_k	-	-
velev-6pipe_k	46.57	**19.46**
velev-10pipe_k	-	-
velev-7pipe_k	-	**2721.39**
velev-9pipe_k	874.26	**232.91**
velev-13pipe_k	-	-

TABLE 6.1 – Zoom sur quelques familles d'instances industrielles

Chapitre 7

Autours des Stratégies de Réduction de la Base de Clauses Apprises

Sommaire

DANS CE CHAPITRE, nous revisitons un point important des solveurs SAT de type CDCL, à savoir les stratégies de gestion de la base des clauses apprises. Notre motivation prend sa source d'une simple observation sur les performances remarquables de deux stratégies simples : élimination des clauses de manière aléatoire et celle utilisant la taille des clauses comme critère pour juger de la pertinence d'une clause apprise. Nous proposons d'abord une stratégie de réduction, appelée "Size-Bounded Randomized strategy" (SBR), qui combine le maintien de clauses courtes (de taille bornée par k), tout en supprimant aléatoirement les clauses de longueurs supérieures à k. La stratégie résultante surpasse celles de état-de-l'art, à savoir la stratégie LBD, sur les instances SAT de la dernière compétition internationale. Renforcé par l'intérêt de garder les clauses courtes, nous proposons plusieurs nouvelles variantes dynamiques, et nous discutons de leurs performances.

7.1 Introduction

Comme nous l'avons souligné dans les chapitres précédents, l'efficacité des solveurs de type CDCL dépend fortement de la stratégie utilisée pour gérer la base des clauses apprises. En effet, comme à chaque conflit une nouvelle clause est ajoutée à la base des clauses apprises, la taille de la base croît de manière exponentielle. Pour éviter cette explosion combinatoire, plusieurs stratégies de gestion de la base des clauses apprises ont été proposées (nous pouvons citer Moskewicz *et al.* (2001), Eén et Sörenson (2004), Audemard et Simon (2009b), Audemard *et al.* (2011), Guo *et al.* (2013)). Ces stratégies visent à maintenir une base de clauses apprises de taille raisonnable en éliminant les clauses jugées non-pertinentes pour la suite de la recherche. Toutes ces stratégies de gestion suivent une séquence temporelle de nettoyage prédéfinie où l'intervalle entre deux étapes successives de réduction est plus ou moins important. A chaque conflit, une activité est associée à la clause apprise (stratégie statique). Une telle heuristique basée sur l'activité vise à pondérer chaque clause en fonction de sa pertinence dans le processus de recherche. Dans le cas des stratégies dynamiques, ces activités sont mises à jour dynamiquement. La réduction de la base des clauses apprises consiste à supprimer les clauses inactives ou non-pertinentes. Même si toutes les stratégies d'élimination des clauses apprises proposées s'avèrent empiriquement efficaces, déterminer la clause la plus pertinente pour le processus de recherche reste un véritable défi. Il est important de noter que l'efficacité de la plupart des stratégies de gestion des clauses apprises de l'état de l'art dépend fortement de la fréquence de nettoyage et de la quantité de clauses à supprimer à chaque nettoyage.

Différentes implémentations de solveurs SAT sont proposées chaque année à la compétition SAT, ils incluent plusieurs améliorations des principales composantes des solveurs CDCL. Les compétitions SAT stimulent le développement des implémentations efficaces. Cependant, une telle course à la mise en œuvre la plus efficace a conduit à rendre les solveurs de plus en plus complexes et dépendants de nombreux paramètres. Ces paramètres sont soient statiques (fixés avant la recherche), soit dynamiques, leurs valeurs sont conditionnellement définies au cours de la recherche en fonction du comportement des solveurs lors de la recherche. Ces implémentations sophistiquées augmentent la difficulté à comprendre ce qui est essentiel de ce qui ne l'est pas. Dans Katebi *et al.* (2011), une analyse empirique portant sur les principales techniques qui ont contribué aux performances impressionnantes des solveurs SAT modernes a été menée. Ceci peut être vu comme un premier pas vers une compréhension profonde des solveurs SAT modernes.

Dans ce chapitre, nous réexaminons une question importante des solveurs SAT de type CDCL, à savoir les stratégies de gestion de la base des clauses apprises. Il est *important de noter* à ce point que, les stratégies qui considèrent les clauses

courtes comme les plus pertinentes ("size-bounded based reduction strategies") ont été proposées dépuis 1996 par Marques Silva et Sakallah (Grasp Marques-Silva et Sakallah (1996a)), Bayardo et Schrag (RelSAT Bayardo Jr. et Schrag (1997)). La plupart des solveurs SAT de l'état de l'art gardent systématiquement uniquement les clauses binaires, tandis que pour les clauses de taille supérieure à deux, plusieurs mesures sophistiquées de pertinence ont été proposées pour prédire la qualité des clauses les plus pertinentes.

Notre motivation pour ce travail prend sa source d'une simple observation sur les performances remarquables des stratégies de réduction basées sur la mise en place d'une borne supérieure sur la taille des clauses apprises. A partir de ces premiers résultats, nous avons décidé d'examiner d'autres stratégies "naïves" telles que la stratégie de gestion aléatoire et celle utilisant une file (First In First Out). Le but est de quantifier l'écart en termes de performance entre ces stratégies simples et celles mises en œuvre dans les solveurs SAT de l'état de l'art. Ensuite, nous dérivons une stratégie de réduction, appelée "Size-Bounded Randomized Strategy" (abrégée SBR), qui conserve les clauses de petites tailles (de taille inférieure ou égale à k), tout en supprimant aléatoirement les clauses de taille supérieure à k. Renforcé par l'intérêt de garder les clauses courtes, nous proposons plusieurs nouvelles mesures dynamiques qui nous permettent de quantifier la pertinence d'une clause apprise donnée en fonction de l'état courant du processus de recherche. Intuitivement, *une clause apprise de petite taille avec les littéraux assignés le plus souvent en haut de la branche courante de l'arbre de recherche est considérée comme plus pertinente*. Plusieurs stratégies basées sur la pertinence sont ensuite dérivées, nous permettant de garder les clauses apprises qui sont plus susceptibles de couper des branches en haut de l'arbre de recherche. Toutes ces stratégies sont intégrées dans le solveur Minisat 2.2 et comparées aux solveurs SAT de l'état de l'art sur les instances d'applications prises de la dernière compétition SAT 2013. Pour confirmer la supériorité des mesures basées sur la taille contre celles basées sur le LBD, nous présentons également les résultats obtenus en substituant LBD avec une activité dynamique basée sur la taille dans le solveur SAT $Glucose\ 3.0$. Les résultats obtenus remettent au goût du jour les anciennes stratégies basées sur la taille des clauses, proposées il y a plus d'une quinzaine d'années Marques-Silva et Sakallah (1996a), Bayardo et Miranker (1996), Bayardo Jr. et Schrag (1997) (voir Section 7.2 - Travaux Connexes). Ces résultats ne sont pas surprenants, puisque les petites clauses sont généralement préférées pour leur capacité à réduire encore plus l'espace de recherche. C'est aussi pourquoi tous les solveurs SAT gardent systématiquement les clauses binaires apprises.

Ce chapitre est organisé comme suit. Nous rappelons tout d'abord les travaux liés dans la section 7.2, ensuite nous motivons notre étude en examinant les stratégies de suppression basées sur la limitation de la taille, sur un choix aléatoire des

clauses à supprimer, et sur une gestion par une file d'attente de la base des clauses ("FIFO") dans la section 7.4. Dans la section 7.5, nous présentons notre première stratégie de réduction appelée stratégie aléatoire bornée sur la taille ("Size-Bounded Randomized strategy"). Dans la section 7.6, nous décrivons plusieurs stratégies de suppression qui nous permettent de garder les clauses apprises qui sont plus susceptibles de couper des branches en haut de l'arbre de recherche. Toutes ces stratégies sont intégrées dans la solveur Minisat 2.2 et comparées aux solveurs SAT de l'état de l'art $Glucose$ 3.0 et $Lingeling$ 2013. Nous présentons également les résultats obtenus en intégrant les stratégies basées sur la taille bornée dans $Glucose$ 3.0. Avant de conclure, une discussion est fournie à la section 7.7.

Les travaux présentés dans ce chapitre ont donné lieu à la publication Jabbour *et al.* (2014c).

7.2 Travaux Connexes

Dans cette section, nous décrivons certaines stratégies de nettoyage de la base des clauses apprises telles que décrites par les auteurs. A notre connaissance, la stratégie visant à maintenir un ensemble pertinent de clauses remonte aux développements de procédures de preuves efficaces basées sur la résolution en démonstration automatique. Dans Gelperin (1973), D. Gelperin propose différentes stratégies qui tentent de determiner la satisfiabilité d'un ensemble de clauses tout en minimisant en même temps la taille de l'ensemble des clauses retenues. Dans les problèmes de satisfaction de contraintes et de satisfiabilité propositionnelle, pour surmonter le surcoût d'apprentissage sans restriction, plusieurs stratégies ont été proposées par Bayardo et al. Bayardo Jr. et Schrag (1997), Bayardo et Miranker (1996), y compris celles d'apprentissages par taille bornée ("Bounded Learning") et par relevance ("Relevance Based Learning"). Ils ont défini la taille bornée (respectivement, la relevance) d'ordre i, qui conserve indéfiniment uniquement les clauses raisons dérivées contenant au plus i variables (respectivement, maintient toute clause raison dérivée qui contient au plus i variables dont les affectations ont changé depuis que la raison a été dérivée). Cette question a d'abord été abordée dans GRASP par Joao Marques Silva et al. Marques-Silva et Sakallah (1996b). En effet, afin de garantir que la croissance de la base des clauses soit polynomiale en nombre de variables dans le pire des cas, dans Marques-Silva et Sakallah (1996b) les auteurs proposent une stratégie selective sur les clauses qui doivent être ajoutées à la base de clauses. Plus précisément, étant donné un paramètre entier k, les clauses conflits dont la taille (nombre de littéraux) est inférieure à k sont marquées en vert (ajoutées à la base des clauses) tandis que celles de taille supérieure à k sont marquées en rouge et conservées autour uniquement lorsqu'elles sont des clauses unitaires, c'est-à-dire,

une clause rouge est supprimée lorsqu'elle contient plus d'un littéral libre (non assigné).

Comme beaucoup d'autres solveurs, Chaff Moskewicz *et al.* (2001) intègre la suppression des clauses conflits ajoutées pour éviter une explosion de la mémoire. Chaff utilise essentiellement les stratégies de suppression de clauses paresseuses planifiées. Lorsqu'une clause est ajoutée, elle est analysée afin de déterminer à quel moment dans l'avenir, le cas échéant, la clause doit être supprimée. La mesure utilisée est la pertinence, de telle sorte que lorsque plus de n (où n est typiquement 100-200) littéraux dans la clause deviendront non-assignés pour la première fois, la clause sera marquée comme étant supprimée. La mémoire réelle associée aux clauses supprimées est récupérée régulièrement avec une étape de compactage.

Dans Berkmin Goldberg et Novikov (2007), les auteurs considèrent que les clauses les plus récemment déduites sont plus précieuses parce qu'il a fallu plus de temps pour les déduire de l'ensemble des clauses originales. La base des clauses est considérée comme une file d'attente ("First In First Out"). La stratégie de suppression de Berkmin maintient les clauses de petites tailles (de taille inférieure à 8) combinée avec la représentation en file d'attente de la base des clauses apprises.

Minisat Eén et Sörenson (2004) supprime agressivement les clauses apprises sur la base d'une activité heuristique similaire à celle des variables. Une clause apprise est considérée comme non pertinente si son activité ou sa participation à l'analyse de conflits récente est marginale Eén et Sörenson (2004). La limite sur le nombre de clauses apprises autorisées augmente après chaque redémarrage. Cette stratégie a été améliorée dans MiniSAT 2.2.

Plus récemment, Audemard and Simon Audemard et Simon (2009b) utilise le nombre de niveaux différents ("LBD - Literal Block Distance") impliqués dans une clause apprise pour quantifier la qualité des clauses apprises. Cette mesure a été identifiée et utilisée dans Audemard *et al.* (2008c) pour prouver l'optimalité du schéma First UIP. Dans Audemard et Simon (2009b) les clauses ayant une plus petite valeur de LBD sont considérées comme étant plus pertinentes. La mesure LBD est intégrée au solveur MiniSAT conduisant au solveur Glucose, l'un des solveurs SAT de l'état de l'art. La mesure LBD est également exploitée dans les solveurs Lingeling Biere (2012), SAT13 Knuth le solveur basé sur CDCL conçu par D. Knuth, et quelques autres solveurs SAT introduits dans la dernière compétition SAT 2013 (exemples $gluebit\_clasp$ 1.0, $BreakIDGlucose$ 1, $glueminisat$ 2.2.7j). Une autre stratégie de gestion dynamique de la base des clauses apprises est proposée dans Audemard *et al.* (2011). Elle est basée sur un principe de gel et d'activation dynamique des clauses apprises. À un état de la recherche donné, en utilisant une fonction de sélection pertinente basée sur une sauvegarde progressive des affectations des variables Pipatsrisawat et Darwiche (2007), elle active les clauses apprises

les plus prometteuses tout en gelant celles jugées non pertinentes.

Dans Guo *et al.* (2013), les auteurs proposent deux mesures pour prédire la qualité des clauses apprises. La première mesure est basée sur le niveau de retour-arrière (BTL), tandis que la seconde est basée sur une notion de distance, définie comme la différence entre le maximum et le minimum des niveaux d'affectations des littéraux impliqués dans la clause apprise.

Enfin, les stratégies d'échange de clauses par taille bornée sont également considérées dans plusieurs solveurs SAT parallèles basés sur le portfolio et diviser pour régner (par exemple ManySAT Hamadi *et al.* (2009c) et PMSat Gil *et al.* (2008)).

7.3 Mise en Contexte

Un solveur SAT basé sur l'apprentissage de clause par analyse de conflit (CDCL) explore l'espace de recherche en effectuant successivement une séquence de décision/propagations. Quand un conflit est rencontré, une clause conflit est dérivée par résolution sur les clauses impliquées dans le processus de propagation unitaire (codé comme un graphe d'implications). Une telle clause conflit apprise est alors ajoutée à la base des clauses apprises. Elle nous permet de produire un littéral unitaire (assertif) à un niveau plus haut de l'arbre de recherche. Ensuite, le solveur SAT basé sur l'architecture CDCL effectue un retour-arrière non chronologique à ce niveau, propage le littéral assertif et répète la séquence de décison/propagations jusqu'à ce qu'un modèle soit trouvé ou alors une clause vide est dérivée. La recherche redémarre régulièrement, et la base des clauses apprises est régulièrement réduite par l'élimination des clauses non pertinentes. Ces différentes composantes sont liées entre elles. Par exemple le redémarrage a des effets très forts sur la composante d'apprentissage Huang (2007), Biere (2008a), Pipatsrisawat et Darwiche (2009b;a), Hamadi *et al.* (2009c), Audemard et Simon (2012). Certains solveurs SAT de l'état de l'art comme $Glucose$ utilisent un redémarrage agressif, très utile pour la résolution des instances insatisfiables.

Dans la suite de ce chapitre, étant donné une clause c, nous définissons c^i comme la projection de c sur les littéraux de c assignés au niveau i, c'est à dire, $c^i = \{\ell | niv(\ell) = i\}$. Cet ensemble est appelé bloc dans Audemard et Simon (2009b).

7.4 Quelques Stratégies de Suppression des Clauses

Comme mentionné dans l'introduction, notre principal objectif est de quantifier tout d'abord l'écart de performance entre les stratégies de suppression des clauses

apprises de l'état de l'art et quelques stratégies basiques, y compris celles basées sur un choix aléatoire des clauses à supprimer. Dans cette section, nous illustrons les performances des stratégies de suppression basées sur la taille, sur une suppression aléatoire ("Random") et sur une gestion en file d'attente de la base des clauses apprises ("FIFO"). Toutes ces stratégies statiques sont intégrées sans aucune autre modification à MiniSAT 2.2. A chaque nettoyage de la base des clauses apprises, nous maintenons uniquement la moitié des clauses de la base estimées comme plus importantes, l'autre moitié considérée comme moins importante est supprimée. Les trois nouvelles versions de MiniSAT sont obtenues comme suit :

- $Size\text{-}MiniSAT$: A chaque conflit, l'activité de la clause apprise c est égale à sa taille, c'est à dire $\mathcal{A}(c) = |c|$
- $Rand\text{-}MiniSAT$: A chaque conflit, l'activité de la clause apprise c est fixée à une value réelle aléatoire $w \in [0..1]$, c'est à dire $\mathcal{A}(c) = drand(random\_seed)$. Nous avons utilisé exactement la fonction aléatoire de MiniSAT avec le même paramètre random_seed pour obtenir des résultats reproductibles.
- $FIFO\text{-}MiniSAT$: Dans cette version, les clauses apprises sont gérées en utilisant une file d'attente. Chaque fois qu'une réduction est effectuée, les clauses les plus anciennes sont supprimées.

$Solveurs$	#Résolues (#SAT - #UNSAT)	Temps moyen
$MiniSAT$	201 (122 - 79)	956.78s
$Glucose3.0$	216 (104 - 112)	**807.62s**
$Lingeling2013$	**233** (119 - **114)**	1090.99s
$Size\text{-}MiniSAT$	221 (**126** - 95)	1233.71s
$Rand\text{-}MiniSAT$	191 (121 - 70)	1071.05s
$FIFO\text{-}MiniSAT$	173 (119 - 54)	862.50s

TABLE 7.1 – Une évaluation comparative des stratégies basées sur Taille/RANDOM/FIFO-MiniSAT

Pour toutes les stratégies de suppression définies dans ce chapitre, les clauses avec les plus petites activités sont considérées comme plus pertinentes. Concernant la fréquence de néttoyage, nous suivons la même stratégie mise en oeuvre dans MiniSAT 2.2. Pour toutes les expérimentations présentées dans ce chapitre, nous exécutons les solveurs SAT sur les 300 instances prises de la compétition SAT 2013. Toutes les instances sont prétraitées par SatElite Eén et Biere (2005) avant l'exécution du solveur SAT. Nous utilisons une machine Quad-Core Intel XEON avec 32GB de mémoire fonctionnant à 2.66 Ghz. Pour chaque instance, nous utilisons un temps limite égal à 5000 secondes du temps CPU.

Dans la table 7.1, nous donnons l'évaluation expérimentale comparative de nos trois versions de $MiniSAT$ avec $Glucose$ 3.0 et $Lingeling$ 2013 le meilleur solveur de la compétition SAT 2013 (sur la catégorie applications). Dans la deuxième colonne, nous donnons le nombre total d'instances résolues (#Résolues). Nous mentionons également, le nombre d'instances prouvées satisfiables (#SAT) et insatisfiables (#UNSAT) entre parenthèses. La troisième colonne indique le temps CPU moyen en secondes (le temps total sur les instances résolues divisé par le nombre d'instances résolues).

A partir de cette première expérimentation, les solveurs SAT de l'état-de-l'art $Lingeling$ 2013 et $Glucose$ 3.0 résolvent 233 et 216 instances respectivement. Le solveur $MiniSAT$ résout 201 instances. Essayons de commenter les performances des autres versions de $MiniSAT$ sans aucune autre amélioration. Le solveur $Size$-$MiniSAT$ est capable de résoudre 221 instances (5 instances de plus que $Glucose$ 3.0). Mais $Glucose$ 3.0 est meilleur en terme de temps CPU moyen. D'autres observations importantes peuvent être tirées de cette première expérimentation. Sur les instances satisfiables $Size$-$MiniSAT$ est plus performant que tous les autres solveurs, il résout 126 instances. La dernière remarque que l'on peut tirer est le fait que $Rand$-$MiniSAT$ résout plus d'instances satisfiables (121) que les solveurs de l'état-de-l'art $Glucose$ 3.0 et $Lingeling$ 2013. Enfin $FIFO$-$MiniSAT$ est clairement le plus mauvais des solveurs, particulièrement sur les instances insatisfiables (seulement 54 instances sont résolues). Cependant, il reste compétitif sur les instances satisfiables et résout 15 instances de plus que $Glucose$

7.5 Stratégie Randomisée Bornée par la Taille

Les résultats remarquables obtenus dans la section 7.4 par $Size$-$MiniSAT$, où les clauses de petites tailles sont considérées comme étant plus pertinentes sont clairement encourageants. Dans cette section, nous proposons une nouvelle stratégie simple de réduction, appelée "Size-Bounded Randomized strategy" (en court SBR), qui maintient les clauses de petites tailles (de taille inférieure à k), tout en supprimant aléatoirement les clauses de taille plus grande que k. L'un des principaux inconvénients de l'activité basée sur la taille est que les clauses de grandes tailles sont souvent considérées comme non pertinentes. Sur certains problèmes difficiles, une telle sélection drastique des clauses de petite taille pourrait avoir des effets négatifs. Pour pallier ce problème, nous avons introduit une certaine randomisation à l'approche basée sur la taille. Plus précisément, la stratégie SBR est définie comme suit : étant donné une borne supérieure k sur la longueur de la clause apprise, chaque fois qu'une clause apprise c est dérivée, si sa taille est plus petite ou égale à k alors $\mathcal{A}(c) = |c|$, sinon $\mathcal{A}(c) = k + drand(random\_seed)$. En d'autres termes nous pré-

férons toujours les clauses courtes, tandis que les clauses de taille plus grande que k sont considérées comme ayant la même taille k avec une valeur aléatoire supplémentaire. De cette manière, les clauses de grandes tailles peuvent être sélectionnées de manière aléatoire. Le solveur $SBR(k)$-$MiniSAT$ (où k est la borne supérieure de la taille) implémente une telle stratégie. Pour déterminer la meilleure borne supérieure de la taille k, nous exécutons $SBR(k)$-$MiniSAT$ avec $k = 2, 5, 10, 12$ et 15.

Solveurs	#Résolues (#SAT - #UNSAT)	Temps moyen
$Glucose$ 3.0	216 (104 - 112)	**807.62s**
$Lingeling$ 2013	233 (119 - **114)**	1090.99s
$SBR(2)$-$MiniSAT$	196 (122 - 74)	1064.17s
$SBR(5)$-$MiniSAT$	218 (118 - 100)	1213.36s
$SBR(10)$-$MiniSAT$	231 (124 - 107)	1158.25s
$SBR(12)$-$MiniSAT$	**239** (**129** - 110)	1265.96s
$SBR(15)$-$MiniSAT$	228 (120 - 108)	1226.96s

TABLE 7.2 – Une évaluation comparative de $SBR(k)$-$MiniSAT$

Les résultats sont présentés dans la table 7.2. $SBR(12)$-$MiniSAT$ obtient les meilleures performances. Elle résout 239 instances, 6 instances de plus que $Lingeling$ 2013 et 23 instances de plus que $Glucose$ 3.0. Ces résultats remarquables remettent au goût du jour les anciennes stratégies basées sur la taille bornée, proposées il y a plus d'une quinzaine d'années Marques-Silva et Sakallah (1996a), Bayardo et Miranker (1996), Bayardo Jr. et Schrag (1997). Les résultats montrent également que l'ajout d'une certaine randomisation pour permettre la selection des clauses de grande taille nous permet d'introduire une certaine diversification à la dérivation par résolution des solveurs SAT de type CDCL. Il est important de noter que notre implémentation proposée peut être obtenue en ajoutant trois lignes de codes à $MiniSAT$ 2.2 sans aucun réglage ou améliorations supplémentaires. Il convient de signaler que les solveurs $SBR(k)$-$MiniSAT$, pour $k = 5, 10, 12, 15$, sont plus performants que le solveur $Glucose$ 3.0.

7.6 Vers des Stratégies de Suppression Basées sur la Pertinence

Renforcé par l'intérêt de garder les clauses de petites tailles, notre but dans cette section est de montrer que d'autres variantes de ces stratégies basées sur la taille

peuvent conduire à une meilleure performance. Plus précisement, notre objectif est de concevoir des stratégies de suppression dynamiques, où les activités des clauses apprises sont mises à jour au cours de la recherche. Dans la plupart des solveurs SAT de l'état de l'art, les activités des clauses apprises sont mises à jour dynamiquement.

7.6.1 Mesures Dynamiques de Pertinence des Clauses

Expliquons l'idée qui nous a guidée dans la conception de ces nouvelles variantes dynamiques. Intuitivement, *une clause apprise de petite taille avec les littéraux assignés le plus souvent au sommet de l'arbre de recherche courant est considérée comme étant plus pertinente*.

Étant donné une interprétation partielle ρ et c une clause de la base des clauses apprises, notre première stratégie dynamique de suppression basée sur la taille est définie comme suit : L'activité initiale de c est égale à $|c|$. Supposons maintenant que $l \in \rho$ est assigné par propagation unitaire au niveau d, grâce à la clause apprise c. Dans ce cas la nouvelle activité de c est égale à d si $d < |c|$. Cette mesure nous permet de garder les clauses apprises qui sont plus susceptibles de couper les branches au sommet de l'arbre de recherche. Par exemple, pour deux clauses de taille égale, nous préférons celles contenant les littéraux assignés au sommet de l'arbre de recherche. En intégrant cette mesure dans MiniSAT 2.2, nous obtenons notre première variante nommée "$SizeD\text{-}MiniSAT$".

Suivant la même idée, nous proposons une seconde version nommée "$Size(k)D\text{-}MiniSAT$". Le but est d'introduire un seuil k sur la longueur des clauses, afin de mettre à jour uniquement l'activité des clauses ayant une activité supérieure à k, et en gardant statique l'activité des clauses courtes (clauses dont la taille est inférieure à k). Plus précisement, l'activité initiale de c est définie comme suit : si $|c| \leq k$ alors $\mathcal{A}(c) = |c|$, sinon $\mathcal{A}(c) = k + |c|$. Chaque fois qu'une clause apprise c est la raison d'un littéral propagé par propagation unitaire au niveau de décison d, son activité est mise à jour comme suit : si $k + d < \mathcal{A}(c)$ alors $\mathcal{A}(c) = k + d$. De cette façon, l'activité d'une clause de taille inférieure à k n'est pas mise à jour.

La troisième variante est définie comme suit : Soit ρ une interprétation partielle conduisant à un conflit au niveau de décision d et c sa clause apprise associée. L'activité de c est initialement fixée à $\mathcal{A}(c) = \sum_{i=1}^{d} i \times |c^i|$. Au cours de la recherche, chaque fois que c est la raison de la propagation d'un littéral, et si sa nouvelle activité par rapport à l'interprétation courante est meilleure, l'activité de c est mise à jour. Une clause c est considérée meilleure qu'une clause c' si $\mathcal{A}(c) < \mathcal{A}(c')$. Cette nouvelle stratégie nous conduit à un nouveau solveur nommé "$RelD\text{-}MiniSAT$".

Dans la table 7.3, nous présentons les résultats comparatifs des trois nouvelles stratégies de suppression dynamiques basées sur la pertinence. Comme nous pou-

$Solveurs$	#Résolues (#SAT - #UNSAT)	Temps moyen
$Glucose$ 3.0	216 (104 - 112)	**807.62s**
$Lingeling$ 2013	**233** (119 - **114)**	1090.99s
$Size(12)D$-$MiniSAT$	**233** (**123** - 110)	1153.9s
$SizeD$-$MiniSAT$	231 (120 - 111)	1174.97s
$RelD$-$MiniSAT$	222 (122 - 100)	1157.31s

TABLE 7.3 – Une évaluation comparative des stratégies de suppression dynamiques.

vons voir, $Lingeling$ 2013 et $Size(12)D$-$MiniSAT$ présentent des performances comparables (233 instances résolues). Nous observons aussi que $Lingeling$ 2013 est meilleur (respectivement mauvais) que $Size(12)D$-$MiniSAT$ sur les instances insatisfiables (respectivement satisfiables). Le solveur MiniSAT intégrant nos trois stratégies de suppression basées sur la pertinence est plus performant que le solveur $Glucose$ 3.0.

Dans cette section, trois stratégies dynamiques sont proposées. Dans ces stratégies, les activités des clauses apprises sont mises à jour dynamiquement durant la recherche. En résumé, la stratégie randomisée bornée par la taille $SBR(12)$-$MiniSAT$ demeure la meilleure, puisqu'elle résout plus d'instances (239 instances résolues) que toutes les stratégies proposées dans ce travail, y compris celles des solveurs SAT de l'état-de-l'art $Lingeling$ 2013 et $Glucose$ 3.0.

Toutes les stratégies proposées peuvent être facilement intégrées à MiniSAT en utilisant quelques lignes de codes.

7.6.2 Substituer le LBD avec la Taille de la Clause dans Glucose

Dans les sections précédentes, nous avons comparé nos stratégies de suppression intégrées à MiniSAT avec les solveurs SAT de l'état de l'art tels que $Glucose$ 3.0 et $Lingeling$ 2013. Ces deux solveurs exploitent la mesure basée sur le LBD pour la gestion de la base des clauses apprises. La question à laquelle nous souhaitons répondre est la suivante : qu'en est-il de la substitution de la mesure LBD par la taille de la clause dans $Glucose$ 3.0 et $Lingeling$ 2013 ? Comme ces deux solveurs exploitent les mesures basées sur le LBD pour la gestion de la base des clauses apprises, nous ne modifions que $Glucose$ 3.0 pour évaluer l'effet de la substitution du LBD avec la taille dynamique des clauses ("$Size(k)D$") présentée dans la section 7.6.1. Le solveur SAT obtenu est appelé "$Size(k)D$-$Glucose$ 3.0", où k est un $seuil$ sur la taille de la clause. Ce nouveau solveur est obtenu en substituant la valeur du LBD par la taille de la clause dans les deux parties suivantes de $Glucose$ 3.0 :

– *Activité initiale de la clause apprise* : Quand une clause apprise c est dérivée, son activité est initialisée comme suit : $\mathcal{A}(c) = |c|$ si $|c| \leq k$, sinon $\mathcal{A}(c) = k + |c|$.
– *Mise-à-jour dynamique de l'activité de la clause* : à chaque conflit, l'activité de chacune des clauses apprises impliquées dans la dérivation de la clause assertive est mise-à-jour comme suit : soit d le niveau de conflit. Si $k+d < \mathcal{A}(c)$ alors $\mathcal{A}(c) = k + d$. Évidemment, l'activité d'une clause de taille inférieure à k n'est pas mise-à-jour.

Le solveur $SBR(k)$-$Glucose$ 3.0 intègre la stratégie de suppression de taille bornée randomisée. Il est obtenu comme suit : chaque fois qu'une clause apprise c est dérivée, elle reçoit $\mathcal{A}(c) = |c|$ si $|c| \leq k$, sinon $\mathcal{A}(c) = k+irand(random\_seed, |V_{\mathcal{F}}|)$, où $irand(random\_seed, |V_{\mathcal{F}}|)$ retourne un nombre entre 0 et $|V_{\mathcal{F}}|$. Cette activité reste inchangée durant la recherche.

De la même manière, nous substituons dans le solveur $Glucose$ 3.0 la valeur du LBD par la mesure proposée dans le solveur "$RelD$-$MiniSAT$" présentée dans la section 7.6.1. Le solveur SAT obtenu est appelé $RelD$-$Glucose$ 3.0. Il est obtenu comme suit :

– *Activité initiale de la clause apprise* : Soit ρ une interprétation partiale conduisant à un conflit au niveau de décision d et c sa clause apprise associée. L'activité de c est initialement fixée à $\mathcal{A}(c) = \sum_{i=1}^{d} i \times |c^i|$.
– *Mise-à-jour dynamique de l'activité de la clause* : à chaque conflit, l'activité de chacune des clauses apprises impliquées dans la dérivation de la clause assertive est mise-à-jour comme suit : soit d' le niveau de conflit. Si $\sum_{i=1}^{d'} i \times |c^i| < \mathcal{A}(c)$ alors $\mathcal{A}(c) = \sum_{i=1}^{d'} i \times |c^i|$.

$Solveurs$	#Résolues (#SAT - #UNSAT)	Temps moyen
$Glucose$ 3.0	216 (104 - 112)	**807.62s**
$Size(12)D$-$Glucose$	**224** (103 - **121**)	919.16s
$Size(15)D$-$Glucose$	219 (100 - 119)	944.36s
$SBR(15)$-$Glucose$	222 (**105** - 117)	1001.34s
$RelD$-$Glucose$	**230** (**106** - **124**)	866.03s

TABLE 7.4 – Une évaluation comparative de $Glucose$, $Size(k)D$-$Glucose$, $SBR(k)$-$Glucose$ et $RelD$-$Glucose$.

Cette expérimentation démontre que la taille de la clause est clairement plus pertinente que la mesure LBD/glue. Comme nous pouvons observer, à partir du résultat présenté dans la table 7.4, $Size(12)D$-$Glucose$ 3.0 et $RelD$-$Glucose$ résolvent respectivement 8 instances et 14 instances de plus que $Glucose$ 3.0. Plus intéressant

Instance	(#Var, #Cl)	Statut	Temps
ctl_4291_567_11_unsat	(17850 , 147308)	UNSAT	4431.87
ctl_4291_567_11_unsat_pre	(15232 , 142785)	UNSAT	4253.48
ctl_4291_567_8_unsat	(17850 , 147312)	UNSAT	3562.12
ctl_4291_567_8_unsat_pre	(15232 , 142789)	UNSAT	3407.92
nossum–sha1/22-160/002	(4128 , 126580)	SAT	2897.44
nossum–sha1/23-96/003	(4288 , 132608)	SAT	3005.76

TABLE 7.5 – $SBR(12)$-$MiniSAT$ sur les instances non résolues (Catégorie Application - Compétition SAT 2013).

encore, sur les instances insatisfiables, nos deux solveurs résolvent respectivement 9 instances et 12 instances de plus que $Glucose$ 3.0. Sur les instances satisfiables, les solveurs $Glucose$ 3.0 et $Size(12)D$-$Glucose$ 3.0 présentent un comportement similaire. $Glucose$ 3.0 résout seulement une instance de plus que $Size(12)D$-$Glucose$ 3.0. Sur ces instances satisfiables, le solveur $RelD$-$Glucose$ résout 2 instances de plus que $Glucose$ 3.0. Cette expérimentation nous donne une illustration définitive que la mesure basée sur la taille de la clause est meilleure que celle basée sur le LBD.

7.6.3 Instances non résolues à la dernière compétition SAT 2013

Dans la table 7.5, nous présentons les résultats obtenus par $SBR(12)$-$MiniSAT$ sur les instances ouvertes prises de la compétition SAT 2013 (catégorie application). Pour chaque instance, nous reportons le nom de l'instance ($Instance$), son nombre de variables et de clauses (#Var, #Cl), le temps cpu (en secondes) utilisé par $SBR(12)$-$MiniSAT$ pour résoudre l'instance ($Temps$).

A la dernière compétition SAT 2013 [6], 12 instances de la catégorie application n'ont été résolues par aucun solveur séquentiel en moins de 5000 secondes. Parmi ces instances, 6 sont résolues par notre solveur $SBR(12)$-$MiniSAT$. Les résultats sont présentés dans la Table 7.5. Ces résultats confirment clairement les bonnes performances de notre solveur $SBR(12)$-$MiniSAT$.

6. instances non résolues $http://edacc4.informatik.uni-ulm.de/SC13/experiment/19/unsolved\text{-}instances/$

7.7 Discussion

En résumé, nous démontrons que la taille de la clause reste la meilleure mesure pour quantifier l'utilité d'une clause donnée. La leçon qu'on peut tirer de cette étude est que la prédiction des meilleures clauses à maintenir durant la recherche mérite une enquête plus poussée. A notre avis, la performance de la mesure LBD peut être expliquée par le fait qu'elle est réellement liée à la taille des clauses. En effet, nous avons $2 \leq LBD(c) \leq |c|$. La taille de la clause est une borne supérieure de la mesure LBD. Par exemple, le LBD des clauses binaires est égal à 2. Par conséquent, la stratégie définie dans $Glucose$ favorise dans un certain sens le maintien des clauses de petites tailles. Dans Audemard et Simon (2009b), les auteurs mentionnent que : "*Un bon schéma d'apprentissage devrait ajouter des liens explicites entre des blocs indépendants des littéraux propagés (ou de décision). Si le solveur reste dans le même espace de recherche, une telle clause aidera probablement à réduire le nombre de prochains niveaux de décision au cours du reste de la recherche*". L'intuition donnée par les auteurs pour comprendre l'importance des clauses de LBD égal à 2 est la suivante : "*la variable du dernier niveau de décision sera collée ("glued") avec le bloc de littéraux propagés à un niveau plus haut de l'arbre de recherche. Nous suspectons toutes ces clauses d'être vraiment importantes au cours de la recherche*". Essayons de discuter sur ces affirmations. Premièrement, toute clause apprise ajoute des liens explicites entre les blocs indépendants de littéraux propagés. Deuxièmement, toute clause apprise aide aussi à élaguer l'espace de recherche et donc à réduire le nombre de décisions. La question qu'on se pose est la suivante : soit $c_1 = (x_1 \vee x_2 \vee x_3)$ et $c_2 = (y_1 \vee y_2 \vee, \ldots, \vee y_n)$ avec $n > 3$, $LBD(c_1) = 3$ et $LBD(c_2) = 2$, laquelle est pertinente ? Comme vous pouvez le deviner, notre préférence va à la première clause.

Du côté théorique, la prédiction de la pertinence d'une clause donnée pour la preuve est liée au problème de trouver les plus courtes réfutations par résolution. Ce dernier problème a été demontré NP-Difficile Iwama (1997), et aussi NP-Difficile en approximant avec un facteur constant Alekhnovich *et al.* (1998). Une deuxième mesure importante est la largeur d'une preuve par résolution, définie comme étant le nombre maximal de littéraux dans une clause de la preuve. Dans Ben-Sasson et Wigderson (1999), une relation intéressante est établie entre la largeur de la preuve et la longueur de la preuve. Ces résultats suggèrent que la taille des clauses apprises est un concept fondamental pour la résolution pratique du problème SAT. Sur certaines instances difficiles, on a besoin de maintenir les clauses de grande taille. Notre stratégie aléatoire est inspirée de cette preuve.

Une remarque importante, toutes les stratégies présentées dans ce chapitre peuvent être intégrées facilement (quelques lignes de code) à tout solveur basé sur l'architecture CDCL. Une page web dédiée, incluant les stratégies proposées pour la ré-

duction de la base des clauses apprises intégrées à MiniSAT est actuellement en construction (http ://www.cril.fr/∼sais).

7.8 Conclusion

Dans ce chapitre, nous avons abordé le problème de gestion de la base des clauses apprises. Nous avons démontré que les stratégies d'apprentissage basées sur la taille bornée proposées il y a plus d'une quinzaine d'années Marques-Silva et Sakallah (1996a), Bayardo et Miranker (1996), Bayardo Jr. et Schrag (1997) restent de bonnes mesures pour prédire la qualité des clauses apprises. Nous avons également montré que l'ajout de la randomisation à l'apprentissage en fixant une borne supérieure sur la taille est un bon moyen de parvenir à une diversification contrôlée. Cette stratégie nous permet de privilégier les clauses de petite taille, tout en maintenant une petite fraction de clauses de grande taille nécessaires pour la dérivation des preuves par résolution sur certaines instances. Les résultats expérimentaux, montrent que la stratégie basée sur la taille bornée randomisée intégrée à MiniSAT peut atteindre des performances meilleures que celles des solveurs SAT de l'état de l'art. Convaincu par l'importance des clauses de petites tailles, nous avons proposé plusieurs variantes dynamiques efficaces qui visent à maintenir des clauses courtes contenant des littéraux qui apparaissent le plus souvent en haut de l'arbre de recherche. Notre dernière évaluation montre que la substitution de la mesure LBD par la mesure basée sur la taille dans $Glucose\ 3.0$ mène à une meilleure performance à la fois sur les instances satisfiables et sur les instances insatisfiables.

Conclusions et Perspectives

Conclusions

Dans cette thèse, nous nous somme intéressé à la résolution du problème de la satisfiabilité d'une formule propositionnelle (SAT), qui est un problème central en théorie de la complexité, en intelligence artificielle et en démonstration automatique.

Dans la première partie, nous avons d'abord introduit quelques principes nécessaires à la compréhension de cette thèse. Ensuite après avoir présenté les différentes approches de résolution du problème SAT, nous avons fait une description des différentes composantes des solveurs SAT de type CDCL généralement appelés solveurs SAT modernes en se focalisant sur les composantes essentielles.

Nous avons présenté dans la deuxième partie les différentes contributions que nous avons apportées dans le cadre de la résolution du problème SAT. L'objectif principal de ces contributions est d'élargir la classe des problèmes traitables en pratique.

La première contribution propose une technique originale de substitution dynamique de fonctions booléennes dans une formule booléenne sous forme CNF. Cette approche exploite les fonctions booléennes cachées généralement introduites pendant la phase d'encodage vers CNF, et les utilise pour minimiser les clauses apprises. Notre approche peut être vue comme une façon originale de manipuler les sous-formules représentées par les fonctions booléennes, nous permettant ainsi d'imiter le principe de résolution étendue. L'avantage de cette technique est qu'elle nous permet d'exploiter ces variables auxiliaires au lieu d'en ajouter des variables supplémentaires au cours de la phase de résolution. La substitution des variables d'entrée par les variables de sortie au cours de l'analyse des conflits, est une façon élégante de manipuler ces variables de sortie.

Nous avons ensuite proposé une méthode permettant d'intensifier la recherche dans les solveurs SAT modernes. Cette approche permet de combiner deux des composants clés des solveurs SAT modernes, à savoir le redémarrage et l'heuristique d'ordonnancement des variables. Combiner ces deux composantes permet au solveur de diriger la recherche sur la sous-formule la plus contraignante. Cette recherche basée sur l'intensification est appliquée à chaque redémarrage du solveur SAT. Il collecte les variables rencontrées durant la dernière analyse de conflit en utilisant un ordre prédéfini. Ces variables sont encore sélectionnées en haut de l'arbre de recherche pendant le prochain redémarrage. Ce simple principe d'intensification

donne des améliorations significatives quand il est intégré aux solveurs SAT de l'état de l'art.

Dans la troisième contribution, nous proposons une nouvelle approche pour l'apprentissage des clauses. Cette approche nous permet de dériver de nouvelles clauses Bi-Assertives. Plus précisément, à chaque conflit, le graphe d'implications est traversé séparément à partir des deux littéraux conflits x et $\neg x$ jusqu'au Premier Unique Point d'Implication (First UIP). Au cours de cette traversée, nous dérivons une nouvelle classe de clauses Bi-Assertives qui peuvent conduire à un graphe d'implications plus compact. Plus intéressant, ces nouvelles clauses Bi-Assertives tendent à être plus courtes et induisent plus d'implications que les clauses Bi-Assertives classiques Pipatsrisawat et Darwiche (2009a; 2010). Notre approche utilise une analyse de conflit séparée à travers les deux branches du graphe conflit pour dériver à la fois les nouvelles clauses Bi-Assertives et la clause assertive classique. En effet la clause assertive classique est progressivement générée à partir des nouvelles clauses Bi-Assertives ce qui diffère de la méthode classique de dérivation de la clause assertive. De cette manière, nous n'avons aucun surcoût de recherche additionnel dû à la dérivation de ces nouvelles clauses Bi-Assertives.

Dans la dernière contribution de cette thèse, nous revisitons les stratégies de réduction de la base des clauses apprises. Tout d'abord, nous démontrons que les stratégies d'apprentissage basées sur la fixation d'une borne supérieure sur la taille des clauses apprises proposées il y a plus d'une quinzaine d'années Marques-Silva et Sakallah (1996a), Bayardo et Miranker (1996), Bayardo Jr. et Schrag (1997) restent de bonnes mesures pour prédire la qualité des clauses apprises. Ensuite, Nous montrons que l'ajout de la randomisation à de telles stratégies est un bon moyen de parvenir à une diversification contrôlée. La stratégie résultante nous permet de privilégier les clauses de petite taille, tout en maintenant une petite fraction de clauses de grande taille nécessaires pour dériver des preuves par résolution sur certaines instances SAT. Convaincu par l'importance des clauses de petite taille, nous proposons plusieurs stratégies dynamiques efficaces qui visent à maintenir des clauses de petite taille contenant des littéraux qui apparaissent le plus souvent en haut de l'arbre de recherche. Notre dernière évaluation montre que la substitution de la mesure LBD par la mesure basée sur la taille dans $Glucose$ mène à une meilleure performance à la fois sur les instances satisfiables et sur les instances insatisfiables.

Perspectives

Les travaux qui ont fait l'objet de cette thèse ouvrent plusieurs nouvelles pistes à explorer. La première piste est relative à notre approche de substitution dynamique de fonctions booléennes dans une formule booléenne sous forme CNF. Le travail

envisagé ici consiste à étendre notre approche en visant une substitution partielle. En effet, même si l'ensemble des variables d'entrée apparaît partiellement dans la clause apprise donnée, on peut toujours substituer ces variables par la variable de sortie correspondante. Dans un tel cas, il pourrait être intéressant de calculer la substitution qui maximise la réduction de la taille de la clause apprise. Dans ces travaux, nous nous sommes limiter à la détection et à la substitution des fonctions booléennes "AND" et "OR", cependant il existe d'autres types de fonctions booléennes dans une formule CNF. On pourra alors étendre l'approche proposée afin de considérer aussi d'autres types de fonctions.

Les résultats obtenus grâce à l'intégration de notre stratégie d'intensification dans les solveurs SAT de l'état-de-l'art montre qu'il reste encore des possibilités d'améliorations des heuristiques d'ordonnancement des variables et que des études plus poussées sur le lien entre les redémarrages, les heuristiques d'ordonnancement des variables et l'apprentissage sont nécessaires. Il est clair qu'une combinaison subtile de ces différents composants pourrait améliorer significativement les performances du solveur, de même qu'un agencement naïf pourrait dégrader considérablement ces performances. Alors trouver une bonne combinaison de ces composants est une question difficile qui mérite d'être étudiée. Dans ce contexte, les premiers résultats obtenus par notre principe d'intensification laissent encore entrevoir des investigations plus poussées, à postériori, sur l'analyse du comportement des solveurs SAT en terme de redémarrage et heuristique de branchement. Une autre piste que nous souhaitons explorer concerne le raffinement de l'heuristique VSIDS.

Le nouveau schéma d'apprentissage de clauses proposé pour la dérivation des nouvelles clauses Bi-Assertives ouvre de nombreuses perspectives. Tout d'abord, Nous envisageons de trouver une meilleure stratégie qui nous permettra de gérer plus finement la nouvelle base de clauses Bi-Assertives. En effet, comme le cas de la gestion de la base des clauses assertives, conserver un grand nombre de clauses Bi-Assertives peut altérer l'efficacité de la propagation unitaire, tandis que supprimer trop de clauses peut rendre l'apprentissage inefficace. Par conséquent, il faut trouver de bons critères permettant d'identifier les clauses Bi-Assertives les plus pertinentes. Enfin, nous projetons d'exploiter le graphe d'implications proposé dans Audemard *et al.* (2008a) pour dériver de nouvelles clauses assertives. En effet, l'addition des arcs (appelés arcs inverses) ensemble avec nos clauses Bi-Assertives pourrait conduire aux clauses assertives meilleures en termes de niveau de back-jumping et de taille.

Finalement, la dernière contribution de cette thèse qui s'articule autours des stratégies de réduction de la base des clauses apprises laissent entrevoir de nombreuses perspectives. En effet, nous avons pu voir aux travers des différentes études expérimentales que l'intégration des nouvelles stratégies proposées pour juger la pertinence d'une clause apprise dans les meilleurs solveurs SAT actuels de l'état de

l'art, conduit à une meilleure performance à la fois sur les instances satisfiables et sur les instances insatisfiables. De plus, cette étude nous révèle que les meilleures stratégies sont celles basées sur la limitation d'une borne k sur la taille des clauses. Cette borne étant définie de manière statique, au vu de l'écart de performance entre les différents solveurs obtenus avec différentes bornes et de la diversité entre les problèmes résolus par chacun des solveurs, la mise en place d'une méthode permettant d'ajuster cette borne de manière dynamique au cours de la recherche est un important challenge que nous souhaitons étudier. Au-délà de l'amélioration des différentes stratégies proposées, Nous envisageons également d'utiliser ces stratégies dans le cadre de la résolution parallèle. Actuellement, nous étudions une approche de type portfolio basée sur l'utilisation de nos stratégies sur différentes unités de calcul.

Index

Voici un index

Bibliographie

Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03), volume 2929 de *Lecture Notes in Computer Science*, Santa Margherita Ligure, Italy, mai 2003. Published in 2004. Springer.

Sheldon B. AKERS : Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978. ISSN 0018–9340.

Michael ALEKHNOVICH, Sam BUSS, Shlomo MORAN et Toniann PITASSI : Minimum propositional proof length is np-hard to linearly approximate. *In* Lubos BRIM, Jozef GRUSKA et Jiri ZLATUSKA, éditeurs : *Mathematical Foundations of Computer Science 1998*, volume 1450 de *Lecture Notes in Computer Science*, pages 176–184. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64827-7. URL http://dx.doi.org/10.1007/BFb0055766.

Fadi A. ALOUL, Arathi RAMANI, Igor L. MARKOV et Karem A. SAKALLAH : Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(9):1117–1137, 2003.

Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : A generalized framework for conflict analysis. *In Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, SAT'08, pages 21–27, Berlin, Heidelberg, 2008a. Springer-Verlag.

Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : A generalized framework for conflict analysis. Rapport technique, available on WebSite : http://research.microsoft.com/apps/pubs/default.aspx?id=70552, 2008b.

Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR et Lakhdar SAIS : A generalized framework for conflict analysis. Rapport technique MSR-TR-2008-34, Microsoft Research, Feb 2008c.

Gilles AUDEMARD, George KATSIRELOS et Laurent SIMON : A restriction of extended resolution for clause learning sat solvers. *In AAAI*, 2010.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : On freezing and reactivating learnt clauses. *In Fourteenth International Conference on Theory and Applications of Satisfiability Testing(SAT'11)*, jun 2011.

Gilles AUDEMARD et Laurent SIMON : Glucose : a solver that predicts learnt clauses quality. Rapport technique, 2009a. SAT 2009 Competition Event Booklet, http//www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.

Gilles AUDEMARD et Laurent SIMON : Predicting learnt clauses quality in modern sat solver. *In Twenty-first International Joint Conference on Artificial Intelligence(IJCAI'09)*, pages 399–404, jul 2009b.

Gilles AUDEMARD et Laurent SIMON : Refining restarts strategies for sat and unsat. *In 18thInternational Conference on Principles and Practice of Constraint Programming(CP'2012)*, pages 118–126, 2012.

Fahiem BACCHUS : Enhancing davis putnam with extended binary clause reasoning. *In National Conference on Artificial Intelligence*, pages 613–619, 2002.

Fahiem BACCHUS et Jonathan WINTER : Effective preprocessing with hyper-resolution and equality reduction. *In SAT*, pages 341–355, 2003.

Peter BARTH : A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. Rapport technique, 1995.

Roberto J. BAYARDO et Daniel P. MIRANKER : A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. *In In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

Roberto J. BAYARDO JR. et Robert C. SCHRAG : Using csp look-back techniques to solve real-world sat instances. *In Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence (Rhode Island, USA), juillet 1997.

Paul BEAME, Henry A. KAUTZ et Ashish SABHARWAL : Understanding the power of clause learning. *In IJCAI*, pages 1194–1201, 2003.

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.

E. BEN-SASSON et A. WIGDERSON : Short proofs are narrow-resolution made simple. *In Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 2–, 1999.

Belaid BENHAMOU et Lakhdar SAIS : Tractability through symmetries in propositional calculus. *J. Autom. Reasoning*, 12(1):89–102, 1994.

A. BIERE : Lingeling and friends entering the sat challenge 2012. *In* A. BALINT, A. BELOV, A. DIEPOLD, S. GERBER, M. JARVISALO, et C. Sinz (EDITORS), éditeurs : *Proceedings of SAT Challenge 2012 : Solver and Benchmark Descriptions*, pages 33–34, University of Helsinki, 2012. vol. B-2012-2 of Department of Computer Science Series of Publications B.

Armin BIERE : Adaptive restart strategies for conflict driven sat solvers. *In* Hans KLEINE BÜNING et Xishun ZHAO, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2008*, volume 4996 de *Lecture Notes in Computer Science*, pages 28–33. Springer Berlin / Heidelberg, 2008a.

Armin BIERE : Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(75-97):45, 2008b.

Armin BIERE : Precosat system description. SAT Competition, solver description, 2009.

Armin BIERE, Alessandro CIMATTI, Edmund M. Jr. CLARKE, Masahiro FUJITA et Yunshan ZHU : Symbolic model checking using sat procedures instead of bdds. *Design Automation Conference*, 0:317–320, 1999.

Alain BILLIONNET et Alain SUTTER : An efficient algorithm for the 3-satisfiability problem. *Operations Research Letters*, 12(1):29 – 36, 1992. ISSN 0167-6377.

Laure BRISOUX, Éric GRÉGOIRE et Lakhdar SAIS : Improving backtrack search for sat by means of redundancy. *In ISMIS*, pages 301–309, 1999.

Maurice BRUYNOOGHE : Analysis of dependencies to improve the behaviour of logic programs. *In CADE*, pages 293–305, 1980.

Randal E. BRYANT : Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. ISSN 0360–0300.

Micheal BURO et Hans KLEINE-BÜNING : Report on the sat competition. Rapport technique, University of Paderborn, 1992.

Thierry CASTELL : Computation of Prime Implicates and Prime Implicants by a variant of the Davis and Putnam procedure. *In IEEE International Conference on Tools with Artificial Intelligence TAI'96, Toulouse France, 16/11/96-19/11/96*, pages 428–429, Los Alamitos, California, novembre 1996. IEEE Computer Society Press.

Philippe CHATALIC et Laurent SIMON : Multi-resolution on compressed sets of clauses. *In ICTAI*, pages 2–10, 2000.

Hubie CHEN, Carla GOMES et Bart SELMAN : Formal models of heavy-tailed behavior in combinatorial search. *In* Toby WALSH, éditeur : *Principles and Practice of Constraint Programming - CP 2001*, volume 2239 de *Lecture Notes in Computer Science*, pages 408–421. Springer Berlin / Heidelberg, 2001.

Stephen A. COOK : The complexity of theorem-proving procedures. *In STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

Stephen A. COOK : A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976.

James M. CRAWFORD, Matthew L. GINSBERG, Eugene M. LUKS et Amitabha ROY : Symmetry-breaking predicates for search problems. *In KR*, pages 148–159, 1996.

Nadia CREIGNOU, Sanjeev KHANNA et Madhu SUDAN : *Complexity classifications of boolean constraint satisfaction problems*, volume 7. Society for Industrial Mathematics, 2001.

Mukesh DALAL : Efficient propositional constraint propagation. *In Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI'92)*, pages 409–414, San-Jose, California (USA), 1992.

Martin DAVIS, George LOGEMANN et Donald LOVELAND : A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5: 394–397, 1962.

Martin DAVIS et Hilary PUTNAM : A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

Rina DECHTER : Enhancement schemes for constraint processing : backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41:273–312, January 1990.

Louise DENNIS : An exploration of semantic resolution, 1994.

Gilles DEQUEN et Olivier DUBOIS : kcnfs : An efficient solver for random k-sat formulae. *In Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)* pro (2004), pages 486–501.

Wiliam H. DOWLING et Jean H. GALLIER : Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD et Jacques CARLIER : Sat versus unsat. *In* 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.

Olivier DUBOIS et Yacine BOUFKHAD : From very hard doubly balanced sat formulae to easy unbalanced sat formulae, variations of the satisfiability threshold. *In* J.G. DING-ZHU DU et P. PARDALOS, éditeurs : *Proceedings of the First Worshop on Satisifiability (SAT'96)*, Siena, Italy, mars 1996.

Niklas EÉN et Armin BIERE : Effective preprocessing in sat through variable and clause elimination. *In Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 61–75, St. Andrews, Scottland, juin 2005. Springer.

Niklas EÉN et Niklas SÖRENSON : An extensible sat-solver. *In Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)* pro (2004), pages 333–336.

Imaz ESCALADA : *Optimisation d'algorithmes d'inférence monotone en logique des propositions et du premier ordre*. Thèse doctorat, Université Paul Sabatier, Toulouse, France, décembre 1989.

Pierre FLENER, Justin PEARSON, Meinolf SELLMANN, Pascal Van HENTENRYCK et Magnus ÅGREN : Dynamic structural symmetry breaking for constraint satisfaction problems. *Constraints*, 14(4):506–538, 2009.

J. FRANCO et M. PAULL : Probabilistic analysis of the davis and putnam procedure for solving the satisfiability problem. *Journal of Discrete Applied Mathematics*, 5:77–87, 1983.

John V. FRANCO : Results related to threshold phenomena research in satisfiability : lower bounds. *Theor. Comput. Sci.*, 265(1-2):147–157, 2001.

Jon William FREEMAN : *Improvements to Propositional Satisfiability Search Algorithms*. Ph.d. thesis, University of Pennsylvania, Department of Computer and Information Science, 1995.

Z. GALIL : On the complexity of regular resolution and the davis-putnam procedure. *Theoretical Computer Science*, 4:23–46, 1977.

David GELPERIN : Deletion-directed search in resolution-based proof procedures. *In Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 47–50, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

R. GÉNISSON et P. SIEGEL : A polynomial method for sub-clauses production. *In Proceedings of the sixth international conference on Artificial intelligence : methodology, systems, applications : methodology, systems, applications*, pages 25–34, River Edge, NJ, USA, 1994. World Scientific Publishing Co., Inc.

Ian P. GENT et Barbara M. SMITH : Symmetry breaking in constraint programming. *In ECAI*, pages 599–603, 2000.

Ian P. GENT et Tobby WALSH : Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

Luís GIL, Paulo FLORES et Luís M. SILVEIRA : PMSat : a parallel version of MiniSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:71–98, 2008.

Matthew L. GINSBERG : Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

Allen GOLDBERG : On the complexity of the satisfiability problem. Rapport technique, New York University, 1979.

Eugene GOLDBERG et Yakov NOVIKOV : Berkmin : A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561, 2007.

Eugene P. GOLDBERG et Yakov NOVIKOV : Berkmin : a fast and robust sat-solver. *In In Proceedings of Design Automation and Test in Europe (DATE'02)*, pages 142–149, Paris, 2002.

Carla P. GOMES, Bart SELMAN et Nuno CRATO : Heavy-tailed distributions in combinatorial search. *In CP*, pages 121–135, 1997.

Carla P. GOMES, Bart SELMAN, Nuno CRATO et Henry KAUTZ : Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24:67–100, February 2000.

Carla P. GOMES, Bart SELMAN et Henry KAUTZ : Boosting combinatorial search through randomization. *In 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. ISBN 0-262-51098-7.

Jens GOTTLIEB, Elena MARCHIORI et Claudio ROSSI : Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50, 2002.

Éric GRÉGOIRE, Bertrand MAZURE, Richard OSTROWSKI et Lakhdar SAÏS : Automatic extraction of functional dependencies. *In 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.

Long GUO, Youssef HAMADI, Said JABBOUR et Lakhdar SAIS : Diversification and intensification in parallel sat solving. *In* David COHEN, éditeur : *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 de *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin / Heidelberg, 2010.

Long GUO, Saïd JABBOUR et Lakhdar SAÏS : Stratégies d'élimination des clauses apprises dans les solveurs sat modernes. *In 9ièmes Journées Francophones de Programmation par Contraintes (JFPC'13)*, pages 147–156, Aix en provence, France, 2013.

A. HAKEN : The intractability of resolution. *Theoretical Computer Science*, 39: 297–308, 1985.

Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : Learning for dynamic subsumption. *In Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '09, pages 328–335, Washington, DC, USA, 2009a. IEEE Computer Society.

Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : Lysat : Solver description. SATRACE, solver description, 2009b.

Youssef HAMADI, Said JABBOUR et Lakhdar SAIS : ManySAT : a Parallel SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009c.

Hyojung HAN et Fabio SOMENZI : On-the-fly clause improvement. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 209–222, Berlin, Heidelberg, 2009. Springer-Verlag.

Jin-Kao HAO et Raphaël DORNE : An empirical comparison of two evolutionary methods for satisfiability problems. *In International Conference on Evolutionary Computation*, pages 451–455, 1994.

Marijn HEULE, Matti JÄRVISALO et Armin BIERE : Clause elimination procedures for cnf formulas. *In Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 357–371, Berlin, Heidelberg, 2010. Springer-Verlag.

John N. HOOKER et V. VINAY : Branching rules for satisfiability (extended abstract). *In Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 426–437, London, UK, 1994. Springer-Verlag. ISBN 3-540-58715-2.

Jinbo HUANG : The effect of restarts on the efficiency of clause learning. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2318–2323, Hyderabad, India, janvier 6-16 2007.

Kazuo IWAMA : Complexity of finding short resolution proofs. *In* Igor PRIVARA et Peter RUZICKA, éditeurs : *Mathematical Foundations of Computer Science 1997*, volume 1295 de *Lecture Notes in Computer Science*, pages 309–318. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-63437-9. URL http://dx.doi.org/10.1007/BFb0029974.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Extending resolution by dynamic substitution of boolean functions. *In 24th International Conference on Tools with Artificial Intelligence (ICTAI'12)*, pages 1029–1034, Athens, Greece, 2012a. IEEE Computer Press.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Intensification de la recherche dans les solveurs sat modernes. *In Huitièmes Journées Francophones de Programmation par Contraintes (JFPC'12)*, JFPC12, pages 156–159, Mai 2012b.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Intensification search in modern sat solvers. *In 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 491–492. Springer, 2012c.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Résolution Étendue par substitution dynamique des fonctions booléennes. *In Huitièmes Journées Francophones de Programmation par Contraintes (JFPC'12)*, JFPC12, pages 146–155, Mai 2012d.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Adding new bi-asserting clauses for faster search in modern sat solvers. *In 10th Symposium on Abstraction, Reformulation, and Approximation (SARA'13)*, pages 66–72. AAAI Press, jul 2013.

Saïd JABBOUR, Jerry LONLAC et Lakhdar SAÏS : Nouvelles clauses bi-assertives et leurs intégration dans les solveurs sat modernes. *In Dixièmes Journées Francophones de Programmation par Contraintes (JFPC'14)*, JFPC14, Juin 2014a.

Saïd JABBOUR, Jerry LONLAC, Lakhdar SAÏS et Clémentin Tayou DJAMEGNI : Résolution Étendue par substitution dynamique des fonctions booléennes. *Revue d'Intelligence Artificielle*, 2014b.

Saïd JABBOUR, Jerry LONLAC, Lakhdar SAÏS et Yakoub SALHI : Autours des stratégies de réduction de la base de clauses apprises. *In Dixièmes Journées Francophones de Programmation par Contraintes (JFPC'14)*, JFPC14, Juin 2014c.

Robert G. JEROSLOW et Jinchang WANG : Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

Matti JÄRVISALO, Armin BIERE et Marijn HEULE : Blocked clause elimination. *In* Javier ESPARZA et Rupak MAJUMDAR, éditeurs : *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 de *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin / Heidelberg, 2010.

Hadi KATEBI, KaremA. SAKALLAH et Joao P. MARQUES-SILVA : Empirical study of the anatomy of modern sat solvers. *In* KaremA. SAKALLAH et Laurent SIMON, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695 de *Lecture Notes in Computer Science*, pages 343–356, 2011. ISBN 978-3-642-21580-3.

H. KAUTZ, E. HORVITZ, Y. RUAN, C. GOMES et B. SELMAN : Dynamic restart policies. *In aaai02*, pages 674–682, 2002.

H.A. KAUTZ, D. MCALLESTER et B. SELMAN : Exploiting variable dependency in local search. *In Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya, Japan, 1997.

Henry KAUTZ et Bart SELMAN : Pushing the envelope : planning, propositional logic, and stochastic search. *In Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, AAAI'96, pages 1194–1201. AAAI Press, 1996.

D. KNUTH : Sat13. http ://www-cs-faculty.stanford.edu/∼uno/programs/sat13.w.

Balakrishnan KRISHNAMURTHY : Shorts proofs for tricky formulas. *Acta Informatica*, 22:253–275, 1985.

Balakrishnan KRISHNAMURTHY et Robert N. MOLL : Examples of hard tautologies in the propositional calculus. *In STOC*, pages 28–37, 1981.

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.

Daniel LE BERRE : Sat4j, un moteur libre de raisonnement en logique propositionnelle, dec 2010.

Chu Min LI et Anbulagan ANBULAGAN : Heuristics based on unit propagation for satisfiability problems. *In Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1*, pages 366–371, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

Chu Min LI et Wen Qi HUANG : Diversification and determinism in local search for satisfiability. *In SAT*, pages 158–172, 2005.

Paolo LIBERATORE : On the complexity of choosing the branching literal in dpll. *Artificial Intelligence*, 116:315–326, January 2000. ISSN 0004-3702.

Donalds W. LOVELAND : A linear format for resolution. 125:147–162, 1970.

Michael LUBY, Alistair SINCLAIR et David ZUCKERMAN : Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993. ISSN 0020-0190.

David LUCKHAM : Refinement theorems in resolution theory. 125:163–190, 1970.

Inês LYNCE et João MARQUES-SILVA : Probing-based preprocessing techniques for propositional satisfiability. *In Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '03, pages 105–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2038-3.

João P. MARQUES-SILVA et Karem A. SAKALLAH : Grasp—a new search algorithm for satisfiability. *In ICCAD '96 : Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design* Marques-Silva et Sakallah (1996b), pages 220–227. ISBN 0-8186-7597-7.

João P. MARQUES-SILVA et Karem A. SAKALLAH : Grasp—a new search algorithm for satisfiability. *In ICCAD '96 : Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA, 1996b. IEEE Computer Society. ISBN 0-8186-7597-7.

David A. MCALLESTER : An Outlook on Truth Maintenance. Rapport technique, MIT, 1980.

Michel MINOUX : LTUR : A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, 29(1):1–12, 15 septembre 1988.

Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : engineering an efficient sat solver. *In DAC '01 : Proceedings of the 38th annual Design Automation Conference*, pages 530–535, New York, NY, USA, juin 2001. ACM. ISBN 1-58113-297-2.

Alexander NADEL et Vadim RYVCHIN : Assignment stack shrinking. *In* Ofer STRICHMAN et Stefan SZEIDER, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 de *Lecture Notes in Computer Science*, pages 375–381. Springer Berlin / Heidelberg, 2010.

Alexandre NADEL, Maon GORDON, Amit PATI et Ziad HANA : Eureka-2006 sat solver. SAT-Race, solver description, 2006.

R. OSTROWSKI, É. GRÉGOIRE, B. MAZURE et L. SAÏS : Recovering and exploiting structural knowledge from cnf formulas. *In cp02*, pages 185–199, Ithaca (N.Y.), 2002.

Christos H. PAPADIMITRIOU : *Computational Complexity*. Addison Wesley Pub. Co., 1994.

Cédric PIETTE, Youssef HAMADI et Saïs LAKHDAR : Vivifying propositional clausal formulae. *In Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'03)*, pages 525–529, Patras (Greece), juillet 2008.

Knot PIPATSRISAWAT et Adnan DARWICHE : A lightweight component caching scheme for satisfiability solvers. *In SAT*, pages 294–299, 2007.

Knot PIPATSRISAWAT et Adnan DARWICHE : A new clause learning scheme for efficient unsatisfiability proofs. *In Proceedings of the Twenty-Third American National Conference on Artificial Intelligence (AAAI'08)*, pages 1481–1484, 2008a.

Knot PIPATSRISAWAT et Adnan DARWICHE : A new clause learning scheme for efficient unsatisfiability proofs. *In AAAI*, pages 1481–1484, 2008b.

Knot PIPATSRISAWAT et Adnan DARWICHE : On the power of clause-learning sat solvers with restarts. *In CP*, pages 654–668, 2009a.

Knot PIPATSRISAWAT et Adnan DARWICHE : Width-based restart policies for clause-learning satisfiability solvers. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 341–355, Berlin, Heidelberg, 2009b. Springer-Verlag. ISBN 978-3-642-02776-5.

Knot PIPATSRISAWAT et Adnan DARWICHE : On modern clause-learning satisfiability solvers. *Journal of Automated Reasoning*, 44(3):277–301, 2010.

Daniele PRETOLANI : *Satisfiability and Hypergraphs*. Thèse de doctorat, dipartimento di Informatica : Universita di Pisa, Genova, Italia, mars 1993. TD-12/93.

Daniele PRETOLANI : Efficiency and stability of hypergraph sat algorithms. *Cliques, coloring, and satisfiability : second DIMACS implementation challenge, October 11-13, 1993*, page 479, 1996.

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

William V. QUINE : *Methods of logics*. Henry Holt, 1950.

A. RAUZY, Lakhdar SAÏS et Laure BRISOUX : Calcul propositionnel : vers une extension du formalisme. *In Actes des Cinquièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-complets(JNPC'99)*, pages 189–198, Lyon, France, jun 1999.

Antoine RAUZY : Polynomial restrictions of sat : What can be done with an efficient implementation of the davis and putnam's procedure. *In* U. MONTANARI et F. ROSSI, éditeurs : *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, volume 976 de *Lecture Notes in Computer Science*, pages 515–532, Cassis, France, septembre 1995. Springer.

Irina RISH et Rina DECHTER : Resolution versus search : Two strategies for sat. *J. Autom. Reasoning*, 24(1/2):225–275, 2000.

John Alan ROBINSON : A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

John Alan ROBINSON : Automatic deduction with hyperresolution. *In Automation of Reasoning-Classical Papers on Computational Logic*, volume 1 and 2. Springer, 1983.

Lawrence RYAN : *Efficient algorithms for clause-learning SAT solvers*. Thèse de doctorat, Simon Fraser University, 2004.

Vadim RYVCHIN et Ofer STRICHMAN : Local restarts. *In SAT*, pages 271–276, 2008.

Thomas SCHIEX et Gérard VERFAILLIE : Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3:48–55, 1993.

Bart SELMAN, Hector J. LEVESQUE et David G. MITCHELL : A new method for solving hard satisfiability problems. *In AAAI*, pages 440–446, 1992.

Bart SELMAN, David G. MITCHELL et Hector J. LEVESQUE : Generating hard satisfiability problems. *Artif. Intell.*, 81(1-2):17–29, 1996.

João P. Marques SILVA et Karem A. SAKALLAH : Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

Niklas SÖRENSSON et Armin BIERE : Minimizing learned clauses. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 237–243, Berlin, Heidelberg, 2009. Springer-Verlag.

Niklas SÖRENSSON et Niklas EÉN : Minisat 2.1 and minisat++ 1.0 - sat race 2008 editions. *SAT 2009 competitive events booklet : preliminary version*, page 31, 2009.

Richard M. STALLMAN et Gerald J. SUSSMAN : Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135 – 196, 1977. ISSN 0004-3702.

Sathiamoorthy SUBBARAYAN et Dhiraj PRADHAN : Niver : Non-increasing variable elimination resolution for preprocessing sat instances. *In* Holger HOOS et David MITCHELL, éditeurs : *Theory and Applications of Satisfiability Testing*, volume 3542 de *Lecture Notes in Computer Science*, pages 899–899. Springer Berlin / Heidelberg, 2005.

G.S. TSEITIN : On the complexity of derivations in the propositional calculus. *In* H.A.O. SLESENKO, éditeur : *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

Alan TURING et Jean-Yves GIRARD : *La machine de Turing*. Édition du Seuil, collection source du savoir, 1991.

Tomás E. URIBE et Mark E. STICKEL : Ordered binary decision diagrams and the davis-putnam procedure. *In CCL '94 : Proceedings of the First International Conference on Constraints in Computational Logics*, pages 34–49, London, UK, 1994. Springer-Verlag. ISBN 3-540-58403-X.

Pascal VANDER-SWALMEN : *Aspects parallèles des problèmes de satisfaisabilité*. Thèse doctorat, Université de Reims Champagne-Ardenne en collaboration avec l'Université de Picardie Jules Verne, UFR Sciences Exactes et Naturelles (URCA), UFR des Sciences (UPJV) CReSTIC (URCA), MIS (UPJV), décembre 2009.

Moshe Y VARDI : On p, np, and computational complexity. *Commun. ACM*, 53 (11):5, 2010.

Miroslav N. VELEV et Randal E. BRYANT : Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *J. Symb. Comput.*, 35:73–106, February 2003.

Toby WALSH : Search in a small world. *In Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, pages 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

R. WILLIAMS, C. GOMES et B. SELMAN : On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. *In International Conference on Theory and Applications of Satisfiability Testing, SAT'2003*, pages 222–230, 2003a.

Ryan WILLIAMS, Carla P. GOMES et Bart SELMAN : Backdoors to typical case complexity. *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178, Acapulco, Mexico, août 9-15 2003b.

Nail' K. ZAMOV et V. I. SHARONOV : On a class of strategies for the resolution method (in russian). *Studies in constructive mathematics and mathematical logic. Part III*, 16:54–64, 1969.

Hantao ZHANG : Sato : An efficient prepositional prover. *In* William MCCUNE, éditeur : *Automated Deduction–CADE-14*, volume 1249 de *Lecture Notes in Computer Science*, pages 272–275. Springer Berlin / Heidelberg, 1997.

Lintao ZHANG, Conor F. MADIGAN, Matthew H. MOSKEWICZ et Sharad MALIK : Efficient conflict driven learning in a boolean satisfiability solver. *In ICCAD '01 : Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285, Piscataway, NJ, USA, novembre 2001. IEEE Press. ISBN 0-7803-7249-2.

Lintao ZHANG et Sharad MALIK : The quest for efficient boolean satisfiability solvers. *In CADE-18 : Proceedings of the 18th International Conference on Au-*

tomated Deduction, pages 295–313, London, UK, 2002. Springer-Verlag. ISBN 3-540-43931-5.

Résumé

Dans cette thèse, nous nous intéressons à la résolution du problème de la satisfiabilité propositionnelle (SAT). Ce problème fondamental en théorie de la complexité est aujourd'hui utilisé dans de nombreux domaines comme la planification, la bioinformatique, la vérification de matériels et de logiciels. En dépit d'énormes progrès observés ces dernières années dans la résolution pratique du problème SAT, il existe encore une forte demande d'algorithmes efficaces pouvant permettre de résoudre les problèmes difficiles. C'est dans ce contexte que se situent les différentes contributions apportées par cette thèse. Ces contributions s'attellent principalement autour de deux composants clés des solveurs SAT : l'apprentissage de clauses et les heuristiques de choix de variables de branchement. Premièrement, nous proposons une méthode de résolution permettant d'exploiter les fonctions booléennes cachées généralement introduites lors de la phase d'encodage CNF pour réduire la taille des clauses apprises au cours de la recherche. Ensuite, nous proposons une approche de résolution basée sur le principe d'intensification qui indique les variables sur lesquelles le solveur devrait brancher prioritairement à chaque redémarrage. Ce principe permet ainsi au solveur de diriger la recherche sur la sous-formule booléenne la plus contraignante et de tirer profit du travail de recherche déjà accompli en évitant d'explorer le même sous-espace de recherche plusieurs fois. Dans une troisième contribution, nous proposons un nouveau schéma d'apprentissage de clauses qui permet de dériver une classe particulière de clauses Bi-Assertives et nous montrons que leur exploitation améliore significativement les performances des solveurs SAT CDCL issus de l'état de l'art. Finalement, nous nous sommes intéressés aux principales stratégies de gestion de la base de clauses apprises utilisées dans la littérature. En effet, partant de deux stratégies de réduction simples : élimination des clauses de manière aléatoire et celle utilisant la taille des clauses comme critère pour juger la qualité d'une clause apprise, et motiver par les résultats obtenus à partir de ces stratégies, nous proposons plusieurs nouvelles stratégies efficaces qui combinent le maintien de clauses courtes (de taille bornée par k), tout en supprimant aléatoirement les clauses de longueurs supérieures à k. Ces nouvelles stratégies nous permettent d'identifier les clauses les plus pertinentes pour le processus de recherche.

Mots-clés: Satisfiabilité Propositionnelle (SAT), Solveurs SAT, Fonctions Booléennes, Apprentissage de clauses, clauses Bi-Assertives.

Abstract

In this thesis, we focus on propositional satisfiability problem (SAT). This fundamental problem in complexity theory is now used in many application domains such as planning, bioinformatic, hardware and software verification. Despite enormous progress observed in recent years in practical SAT solving, there is still a strong demand of efficient algorithms that can help to solve hard problems. Our contributions fit in this context. We focus on improving two of the key components of SAT solvers : clause learning and variable ordering heuristics. First, we propose a resolution method that allows to exploit hidden Boolean functions generally introduced during the encoding phase CNF to reduce the size of clauses learned during the search. Then, we propose an resolution approach based on the intensification principle that circumscribe the variables on which the solver should branch in priority at each restart. This principle allows the solver to direct the search to the most constrained sub-formula and takes advantage of the previous search to avoid exploring the same part of the search space several times. In a third contribution, we propose a new clause learning scheme that allows to derive a particular Bi-Asserting clauses and we show that their exploitation significantly improves the performance of the state-of-the art CDCL SAT solvers. Finally, we were interested to the main learned clauses database reduction strategies used in the literature. Indeed, starting from two simple strategies : random and size-bounded reduction strategies, and motivated by the results obtained from these strategies, we proposed several new effective ones that combine maintaing short clauses (of size bounded by k), while deleting randomly clauses of size greater than k. Several other efficient variants are proposed. These new strategies allow us to identify the most important learned clauses for the search process.

Keywords: Propositional Satisfiability (SAT), Boolean Functions, Clauses Learning, Bi-Asserting Clauses.