UNIVERSITÉ D'ARTOIS

Microsoft Research

centre de recherche en informatique de lens

Résolution séquentielle et parallèle du problème de la Satisfiabilité propositionnelle

THÈSE

présentée et soutenue publiquement le 8 Juillet 2013

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Long Guo

Composition du jury

Rapporteurs :	Chu Min Li	Université de Picardie
	Michaël Krajecki	Université de Reims
Examinateurs :	Belaïd Benhamou	Université de Provence
	Fred Hemery	Université d'Artois
Co-Directeurs :	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 tiens à remercier en premier lieu Monsieur Michaël Krajecki et Monsieur Chu Min Li pour avoir accepté de rapporter ce travail, malgré leurs nombreuses activités et responsabilités. je remercie également Monsieur Belaïd Benhamou et Monsieur Fred Hemery pour avoir accepté de siéger dans ce jury en qualité d'examinateurs.

Je remercie infiniment mon directeur de thèse Monsieur Lakhdar Saïs et mon co-directeur de thèse Monsieur Said Jabbour , qui m'ont engagé et dirigé sur la voie de la recherche et sur la voie de la vie.

Je remercie également Monsieur Youssef Hamadi et le Microsoft Research pour leur soutien financier, très important pour mener à bien ce travail.

Merci à tous les membres du CRIL pour leurs disponibilités et pour la bonne ambiance régnante au sein du laboratoire ainsi qu'à tous les doctorant et les docteurs avec qui j'ai passé de bons moments des discussion et à jouer au ping-pong.

Merci à mes parents, leur soutiens inconditionnel m'accompagnent toute la vie. Ils m'ont encouragé tout au long de cette thèse.

Merci à ma femme et ma fille, sans elles ma thèse serait finie un an plus tôt mais il aurait sûrment beaucoup moins de sourire dans la vie.

Je dédie cette thèse
à ma famille.

Table des matières

Liste des tableaux

Table des figures

Liste des Algorithmes

Introduction générale

Le problème SAT, pour « *Boolean SATisfiability problem* », est un problème de décision qui vise à décider si une formule mise sous forme normal conjonctive CNF (CONJUNCTIVE NORMAL FORM) admet une valuation sur un ensemble de variables propositionnelles qui la rend vraie. Depuis plusieurs années, des progrès remarquables ont été obtenus sur le double plan de la modélisation et et de la résolution. Sur le plan de la modélisation, de nombreux problèmes issues de divers domaines peuvent maintenant être modélisés sous forme de problèmes SAT. Par exemple, la vérification formelle et plus particulièrement BMC « Bounded Model Checkling (Biere *et al.* (1999a)) a été parmi les premières applications à être encodé en SAT avec le succès que l'on connait. Nous pouvons aussi citer la gestion des dépendances dans les paquets d'une distribution linux (Argelich *et al.* (2010)) ou dans les plugins du logiciel Eclipse (Le Berre et Rapicault (2009)). Après la phase d'encodage en SAT, il est possible d'essayer de résoudre les instances industrielles SAT obtenues en utilisant de nombreux solveurs existant dans la communauté. Prenons l'exemple précédent, à chaque fois qu'un plugin est installé ou mis à jour dans Eclipse, le solveur SAT4J (Le Berre et Parrain) est appelé afin de déterminer s'il n'y a pas de conflits entre les paquets installés et déterminer quels sont les autres plugins nécessaires. En effet, de nouvelles applications issues de domaines variés comme la cryptographie, la configuration de produit, la déduction automatique, la planification et le bio-informatique peuvent aujourd'hui être modélisées et résolues par des approches SAT.

Sur le volet résolution, comme le problème SAT est le premier problème démontré NP-complet (Cook (1971)), son caractère NP-complet reste toujours le verrou théorique majeur. Mais cela n'a pas empêché la mise en œuvre de méthodes de résolution efficaces. Depuis plusieurs années, grâce aux avancées scientifiques dans le domaine, nous avons maintenant une meilleure compréhension de la nature de la difficulté des problèmes (*i.e.* phénomène de longue traîne *heavy tailed* Gomes *et al.* (1997; 2000), ensemble de variables essentielles *backdoor* Williams *et al.* (2003), et *backbone* Dequen et Dubois (2004)), une structure de données plus efficaces (*i.e. watched literals* Zhang (1997), Zhang *et al.* (2001)), 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)), des nouveaux paradigmes algorithmiques (*i.e.* la randomization et les redémarrages (Gomes *et al.* (1998))). La naissance des solveurs SAT modernes (nommés CDCL, pour Conflict Driven Clauses Learning) a permi de résoudre des instances industrielles de plusieurs centaines de milliers de variables et de millions de clauses en seulement quelques minutes. Ces résultats spectaculaires sont accompagnés aussi de problèmes ouverts. Pour s'attaquer à ces problèmes, la résolution parallèle constitue une bonne direction de recherche. L'avénement récent de nouvelles architectures multicœures a accentué l'intérêt vis à vis du parallélisme qui s'est traduit par l'apparition d'une génération de solveurs parallèles (Sinz *et al.* (2001), Chu et Stuckey (2008), Hamadi *et al.* (2009d), Guo *et al.* (2010)).

C'est dans ce contexte que s'inscrit cette thèse. Notre objectif est de proposer de nouvelles approches originales en utilisant des architectures multicœurs d'ordinateurs pour élargir la classe des problèmes SAT que l'on peut résoudre en pratique.

La thèse est divisée en deux parties. Tout d'abord, nous introduisons l'état de l'art du problème SAT. Dans le premier chapitre de la partie état de l'art, nous donnons les bases de la théorie de la complexité ainsi que les définitions essentielles et formelles du problème SAT. Dans le deuxième chapitre, les descriptions des approches principales de résolution du problème SAT seront données. Nous détaillons les différents composants des solveurs SAT modernes : les solveurs CDCL dans le troisième chapitre. Finalement, nous présentons dans le quatrième chapitre les approches parallèles de résolution du problème SAT.

Dans la deuxième partie, nous présentons des contributions s'intégrant dans le cadre de la résolution parallèle et séquentielle du problème SAT. Notre première contribution consiste à trouver un compromis entre la diversification et l'intensification dans le cadre des solveurs SAT parallèles de type portfolio. Il consiste à lancer plusieurs solveurs CDCL en parallèle et de les laisser résoudre en mode collaboratif et compétitif. Pour faire cela, nous avons nous utilisons l'architecture maître/esclave. Le rôle du maitre est de fournir des informations pertinentes à l'esclave pour le guider vers une certaine partie de l'espace de recherche. Nous avons exploré aussi les différentes topologies que nous avons implémenté au dessus de l'architecture de MANYSAT Hamadi *et al.* (2009d).

La seconde contribution s'intègre aussi dans le cadre des solveurs SAT parallèles de type portfolio. Nous proposons d'ajuster dynamiquement la configuration d'un solveur lorsqu'il est établi qu'un autre effectue un travail similaire. Pour cela, nous avons décrit une nouvelle mesure permettant d'estimer si deux unités de calculs explorent de la même manière l'espace de recherche développé.

La troisième contribution consiste en la mise en œuvre d'une stratégie de réduction de la base des clauses apprises. Inspiré par un critère simple mais efficace (garder les clauses apprises aléatoirement). Nous avons proposé une amélioration de la stratégie de réduction de la base des clauses apprises en étudiant deux composantes importantes : les clauses à supprimer et la fréquence de réduction. Nous nous basons ensuite sur ces deux critères pour définir une stratégie de diversification dans un cadre parallèle.

Dans le huitième chapitre, nous présentons une nouvelle approche nommée « contrôle virtuelle » qui a pour objectif de partitionner l'espace de recherche de chaque moteur séquentiel dans un solveur parallèle, en utilisant les chaînes d'équivalences. Cette approche consiste à distribuer des contraintes complémentaires aux moteurs séquentiels d'un solveur parallèle et vérifier leurs cohérences pendant la recherche. Au lieu de forcer le moteur séquentiel à explorer l'espace de recherche comme dans le cas du paradigme diviser pour régner en utilisant les chemins de guidages, notre approche ne force pas les moteurs de résolution mais leur fourni un cadre et les laisse explorer librement l'espace modulo les contraintes définies par des chaînes d'équivalences.

Enfin pour terminer ce manuscrit, une conclusion générale et de nombreuses perspectives ouvrant des voies de recherches future sont présentées.

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 les problème de modèle checking, la déduction automatique, de planification, la bio-informatique *etc.* peuvent être réduits au problème SAT. Depuis plusieurs années, de nombreux algorithmes ont été proposé 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 dans quelque minutes.

Cette partie se compose de quatre chapitres. Le premier chapitre est consacré à la présentation de problème SAT. Pour commencer, nous définissons le formalisme de la logique propositionnel. Une fois le formalisme est défini, nous introduisons quelque notions de complexité afin de situer au mieux le problème SAT et son intérêt théorique. Nous donnons ensuite certaines notations, définitions et propriétés utiles à la compréhension de la suite de ce manuscrit.

Dans le deuxième chapitre, les méthodes de résolution sont introduites, nous présentons les différents méthodes pour résoudre le problème SAT et les composantes qui ont conduit à l'élaboration des prouveurs SAT modernes.

Le troisième chapitre est consacré au prouveur CDCL, nous détaillons les briques importants dans 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.

A la fin de cette partie, un chapitre consacré à la résolution parallèle du problème SAT est introduit. Après avoir rappelé l'intérêt du parallélisme, nous introduisons le concept d'architecture multi-cœurs et les deux types d'approches (*diviser pour régner* et *portfolio*) utilisé afin de résoudre le problème SAT.

Chapitre 1

Problématique, Définitions et Notations

Sommaire

CE CHAPITRE a pour but de fournir aux lecteurs les briques élémentaires de la logique propositionnelle. Dans un premier temps, nous définissons de manière formelle ce qu'est le problème de la satisfiabilité d'une formule propositionnelle (en d'autre mot le problème SAT). Après ces définitions, nous donnons quelques notions de complexité de manière à mieux situer l'importance théorique et pratique de ce problème. Nous terminons ce chapitre par une revue des différents types de problèmes encodé 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 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 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. Afin de mieux comprendre ce problème, nous introduisons d'abord quelques notions de la logique propositionnelle.

1.1.1 Logique propositionnelle

Syntaxe

Nous commençons par définir les principaux éléments syntaxiques de la logique propositionnelle. Pour spécifier la syntaxe, il est nécessaire de définir d'abord les atomes pouvant être utilisés pour for-

muler des énoncés :

Définition 1.1 (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 1.2 (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 1.3 (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 1.4 (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* $\neg\ell$ *ou* $\bar{\ell}$ *le littéral complémentaire de* ℓ*.*

Définition 1.5 (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* $\neg\ell$ *n'apparaît pas dans* $\mathcal{F}$*.*

Nous fixons quelques notations supplémentaires usuelles. 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 1.6 (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 1.1. *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 1.7 (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 1.1. *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 1.8 (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 1.9 (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 1.10 (formule satisfiable)**.** *Une formule propositionnelle $\mathcal{F}$ est satisfiable si elle admet au moins un modèle, i.e., $M(\mathcal{F}) \neq \emptyset$.*

Définition 1.11 (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 1.12 (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 1.13 (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 1.2. *De la définition d'une interprétation on déduit les équivalences suivantes :* $(\mathcal{F} \Rightarrow \mathcal{G}) \equiv \neg\mathcal{F} \vee \mathcal{G}$ *et* $(\mathcal{F} \Leftrightarrow \mathcal{G}) \equiv (\mathcal{F} \Rightarrow \mathcal{G}) \wedge (\mathcal{G} \Rightarrow \mathcal{F})$.

Nous introduisons maintenant un théorème très important qui nous permet d'énoncer un résultat fondamentale en démonstration automatique (preuve par l'absurde).

Théorème 1.1 (déduction)**.** *Soit $\mathcal{F}$ et $\mathcal{G}$ deux formules propositionnels, $\mathcal{F} \models \mathcal{G}$ ssi $\mathcal{F} \wedge \neg\mathcal{G}$ est une formule insatisfiable.*

Ce théorème montre que prouver l'implication sémantique est équivalent à montrer l'inconsistance d'une formule. La plupart des algorithmes de démonstration automatique autour du problème SAT exploitent ce théorème.

Formes normales

Dans cette section nous nous intéressons aux différentes formes normales. Nous commençons par introduire la notion de clause :

Définition 1.14 (clause)**.** *Une clause est une disjonction finie de littéraux. Une clause ne contenant pas de littéraux complémentaires est dit fondamentale, sinon elle est dite tautologique.*

Définition 1.15 (différentes variantes de clauses)**.** *Soit c une clause , elle est dite :*

- (UNITAIRE) *ssi elle contient seulement 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 1.16 (Différents types d'instances)**.** *Soit $\mathcal{F}$ une instance* SAT*, $\mathcal{F}$ est une instance :*

- *k-*SAT *si toutes les 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 les clauses de $\mathcal{F}$ sont Horn (respectivement reverse-horn). En ce qui concerne le test de satisfiabilité d'un ensemble de clauses de Horn ou reverse-horn, plusieurs auteurs ont montré que la résolution est polynomiale voir même linéaire Minoux (1988), Dalal (1992), Rauzy (1995) ;*

Définition 1.17 (terme)**.** *Un terme est une conjonction finie de littéraux.*

Définition 1.18 (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 1.2. *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 est fait référence à une formule propositionnelle, sauf mention contraire, il s'agit d'une formule sous forme CNF.

Propriété 1.2. *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).

1.1.2 le problème SAT

Nous donnons maintenant la définition concrète du problème SAT et quelque règles fondamentales pour simplifier la formule CNF.

Tout d'abord, nous introduisons le problème de décision :

Définition 1.19 (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 résolvants et répondant « Oui » ;*
2. N_Π *: l'ensemble des instances pour lesquelles la réponse est « Non ».*

Définition 1.20 (problème de décision complémentaire)**.** *Soit* Π *un problème de décision, le problème complémentaire* Π^c *de* Π *est un aussi un problème de décision tel que :*

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

De nombreux problèmes informatique peuvent être réduit 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 1.21 (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 1.3. *Soit une formule booléenne mise sous forme* CNF $\mathcal{F} = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a)$*, est-ce que la formule* $\mathcal{F}$ *admet au moins 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 qui permettent de transformer une formule CNF ont été introduite. Ces techniques visent soit à ajouter ou supprimer de la redondance dans les formules CNF.

La première approche présenté ici, c'est sans aucun doute l'une de règles les plus fondamentales de la logique propositionnelle : la règle de résolution.

Définition 1.22 (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 1.4. *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 1.3. *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 :*

- $c_i \wedge c_j \models r$
- $\mathcal{F} \equiv \mathcal{F} \wedge r$

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 1.23 (fusion)**.** *Si* $c_i = \{x_1, x_2, , ...x_n, l, y_1, y_2, ..., y_m, l, z_1, z_2, ..., z_k, l\}$ *est une clause dans une formule* $\mathcal{F}$*, alors* 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 pour supprimer des littéraux dans les clauses. Nous introduisons maintenant deux techniques permettent la suppression de clauses de la formule.

Définition 1.24 (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 1.5. *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 1.25 (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 1.6. *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 1.26 (clause redondante)**.** *Soie* 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 de clauses redondantes, la suppression de clauses redondantes permet de simplifier la formule en préservant sa satisfiabilité.

Le problème SAT est le problème NP-Complet de référence Cook (1971). Pour mieux comprendre son intérêt sur le plan théorique, nous allons présenter des notions de complexité dans la section suivante.

1.2 Théorie de la complexité des algorithmes

Cette section a simplement pour objectif de situer le problème SAT dans la hiérarchie polynomiale. 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 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 dit, 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.

Deux types de complexités sont distingués lors qu'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.

1.2.1 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 du 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 1.7. *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 situe au dernier 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 1.27 (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 tel 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 1.8. *L'algorithme utilisé dans l'exemple 1.7 est de complexité $\mathcal{O}(n)$.*

Définition 1.28 (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 de $\mathcal{O}(n^i)$.*

Exemple 1.9. *Supposons que le nombre d'opération 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 1.29 (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 algorithme 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 être phénoménales. Plus précisément, les problèmes d'une complexité exponentielle ou factorielle sont impossible à résoudre sur l'ensemble des données de taille raisonnable (n>250).

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.

1.2.2 La machine de Turing

Les machines de Turing ne sont pas des machine 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 finie d'états possibles et de transitions qui régissent des calculs de la machine. Les symboles de la bandes 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 la 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 sont décrites par les transitions de la machine.

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

Définition 1.30 (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 1.31 (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 1.32 (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.

1.2.3 Classes de complexité des problèmes de décision

La machine de Turing peut être utilisée pour définir des classes de problèmes selon leur 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 1.33 (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 1.34 (classe NP)**.** *La classe de NP regroupe l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing non-déterministe.*

Définition 1.35 (classe $CoNP$)**.** *La classe $CoNP$ regroupe l'ensemble des problème 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 1.4. *Nous avons : $P \subseteq NP et P \subseteq CoNP$.*

Définition 1.36 (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 1.37 (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 1.3. *Quand on parle de problème C-difficile ou C-complet, on s'autorise uniquement des réduction 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.

1.3 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 le problème SAT sont généralement évaluées sur un ensemble de formule 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. Pour évaluer des performances des différents algorithmes (ou prouveurs) et pour encourager la recherche d'algorithme 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 des expérimentations sont présentées, les instances sont issues de ces différentes compétitions.

1.3.1 Instances aléatoires

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

Définition 1.38 (instance aléatoire k-SAT)**.** *Une instance aléatoire de k-SAT est une formule SAT $\mathcal{F}_k(n, m)$, avec m clauses sur n variables, tous ses clauses sont choisi aléatoirement entre $2^k\binom{n}{k}$ 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.*

Plusieurs générateurs d'instances aléatoires sont proposés Franco (2001), mais le générateur d'instance aléatoire k-SAT est le plus étudié. Dans Selman *et al.* (1996), les auteurs ont donné un constatation que pour $k \geq 3$, il existe un intervalle du rapport entre le nombre de clause et le nombre de variable, $r = \frac{m}{n}$, dans lequel c'est difficile à *décider* si une instance aléatoire de k-SAT et satisfiable ou pas. Nous pouvons alors prévoir la difficulté de tels instances.

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

1.3.2 Instances élaborés

Les instances élaborés 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 leur propriétés mathématiques dont la difficulté est difficile à prévoir

1.3.3 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.* (1999b) et planification Kautz et Selman (1996). Ces instances n'ont pas réellement de caractéristiques permettant de prévoir leur difficultés, mais elles ont souvent leurs structures intrinsèque.

1.4 Conclusion

Dans ce chapitre, nous avons introduit le problème NP-complet de référence : le problème SAT. A cause de sa difficulté théorique, les performances des algorithmes doivent être évalué empiriquement, et le choix des instances (base de tests) est alors très important. Nous avons présenté trois catégories d'instances : les instances aléatoires, les instances élaborés et les instances industrielles. Dans les prochains chapitres nous présentons plusieurs paradigmes pour la résolution de ce problème.

Chapitre 2

Méthodes de résolution

Sommaire

DE NOMBREUX algorithmes ont été proposés pour résoudre SAT. Les principes à la base de ces méthodes sont très variés : résolution Robinson (1965), Galil (1977), énumération Quine (1950), Davis *et al.* (1962), Jeroslow et Wang (1990), 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.

En générale, les méthodes pour la résolution pratique du problème SAT se divisent en trois catégories : les approches incomplètes, les approches complètes et les approches hybrides.

Les techniques basées sur la recherche locale qui est certainement le plus connue de tous les approches incomplètes pour la résolution de problème SAT. Elles ont été initialement développées pour résoudre des problème d'optimisation combinatoire (problème du voyageur de commerce, sat à dos, *etc.*). Elles font partie de la famille des métaheuristiques et sont issues de la recherche opérationnel. Un des inconvénients majeurs de ce type d'approches sont qu'elles sont limités à la recherche d'un modèle sans garantir son obtention (sauf adaptation particulière) même considérant un temps infini. Elles s'appuient sur un parcours non systématique et souvent stochastique de l'espace de recherche. Le principe de base des méthodes de recherche local consiste à se déplacer judicieusement dans l'espace des configurations en améliorant la configuration courante. Dans le cas de SAT, les configurations sont des interprétations complètes de la CNF et l'amélioration se juge en terme de nombre de clause 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 raisonnable. Et elles s'avèrent plus efficace sur les instances aléatoires.

Les algorithmes complets permettent, en un temps fini, de déterminer la consistance de n'importe quelle formule exprimée sous forme CNF. Ils s'appuient généralement sur un parcours en profondeur d'un arbre de recherche, où chaque nœud correspond à l'assignation d'une variable et chaque chemin correspondant à une interprétation partielle des variables de la formule. L'objectif est donc de déterminer un chemin de la racine à une feuille - lequel représente une interprétation complète des variables de la

formule qui satisfait l'ensemble des contraintes du problème ou de déterminer qu'il n'existe pas un tel chemin. Le nombre d'étapes nécessaire pour ce type d'algorithme est proportionnel au nombre d'interprétations possibles, et est donc d'ordre exponentiel en fonction du nombre de variables de la formule. Il est par conséquent quasiment impossible d'obtenir une réponse dans une durée raisonnable si la taille du problème devient trop grande.

L'idée des approches hybrides qui combine les algorithmes de recherche local et les approches complètes est d'exploiter les forces de l'une pour atténuer les faiblesses de l'autre, le but est obtenir un algorithme au moins efficace que les deux approches prises séparément.

Dans ce manuscrit, nous nous concentrons sur les approches complètes. Nous présentons dans ce chapitre une évaluation des méthodes de résolution, de la preuve par résolution au prouveurs SAT moderne.

2.1 Preuve par résolution

Le principe de résolution de Robinson (1965) est l'application la plus fondamentale de la logique propositionnelle. Appliqué à une formule, il nous permet de établir une dérivation :

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 $\pi = [c_1, c_2, ..., c_n = c]$ tel que $\forall i \in (1, k]$, soit (1) $c_i \in \mathcal{F}$, soit (2) $c_k = \eta[x, c_i, c_j]$ avec $1 \leq i, j < k$.*

Toute dérivation par résolution d'une clause vide $(c = \bot)$ à partir de $\mathcal{F}$ est appelée *réfutation* ou *preuve*. Dans ce cas, $\mathcal{F}$ est insatisfiable.

Les trois règles présenté précédemment (voir 1.1.2, 1.1.2, 1.1.2) permettent d'établir une méthode de démonstration automatique complète pour la résolution. La règle de résolution 1.1.2 seul est complète pour la réfutation.

Il est commun de représenter la séquence de dérivation par un graphe de résolution, ce graphe est un DAG (graphe dirigé acyclique).

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 $\mathcal{F} = \{(p \vee \neg r \vee \neg t), (t \vee \neg q), (q \vee r)\}$ et $\alpha = (p \vee t \vee \neg q)$. La Figure 2.1 représente sous forme arborescente la séquence de dérivation $[(p \vee \neg r \vee \neg t), (t \vee \neg q), (p \vee \neg r \vee \neg q), (q \vee r), (t \vee r), (p \vee t \vee \neg q)]$.*

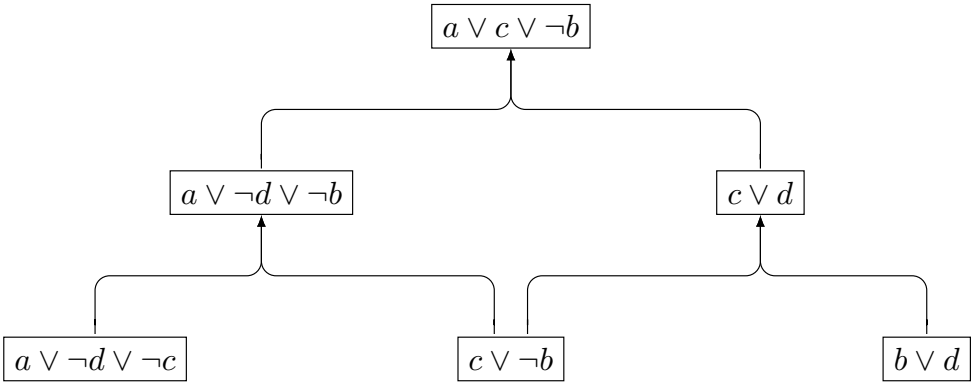


FIGURE 2.1 – Graphe de résolution.

Malgré sa simplicité, la résolution est difficile à mettre en œuvre efficacement, car le nombre de résolvantes à effectuer est souvent exponentiel, l'espace mémoire requis devient alors très élevé et les temps de calculs prohibitifs. De nombreuses restrictions basées sur la structure de la preuve par résolution ont été proposées, elles sont généralement plus facile à mettre en œuvre. Parmi elles, 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'autre 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 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 $y \Leftrightarrow (\ell_1 \vee \ell_2)$ (*i.e.* $(y \vee \ell_1), (y \vee \ell_2)$ et $(\neg y \vee \neg \ell_1 \vee \neg \ell_2)$), 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) ;

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 ;

Une autre méthode 2.1 proposée par **Davis et Putnam** (1960) est l'une des premières méthodes de résolution dédiée au probème SAT. Son principe est basé sur l'application de la résolution comme

Algorithme 2.1 : DP

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

Début
  tant que $(\mathcal{V}_\mathcal{F} \neq \emptyset)$ **faire**
    $x =$`ChoisirVariable`$(\mathcal{V}_\mathcal{F})$;
    $\mathcal{V}_\mathcal{F} = \mathcal{V}_\mathcal{F} - x$;
    $\mathcal{F} \longleftarrow \mathcal{F} - (\mathcal{F}_x \cup \mathcal{F}_{\neg x})$;
    $\mathcal{F} \longleftarrow \mathcal{F} \cup \eta[x, \mathcal{F}_x, \mathcal{F}_{\neg x}]$;
    si $\emptyset \in \mathcal{F}$ **alors retourner** `faux` ;
  retourner `vrai` ;
Fin

principe d'élimination de variables. En effet, étant donnée une variable x, 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 générer toutes les résolvantes possible entre $\mathcal{F}_x$ et $\mathcal{F}_{\neg x}$ noté $\eta[x, \mathcal{F}_x, \mathcal{F}_{\neg x}]$. Les deux sous-ensembles $\mathcal{F}_x$ et $\mathcal{F}_{\neg x}$ sont ensuite omis de $\mathcal{F}$ et remplacés par $\eta[x, \mathcal{F}_x, \mathcal{F}_{\neg x}]$. On note donc, qu'à chaque étape, le sous-problème généré contient une variable en moins, mais un nombre quadratique de clauses supplémentaires. Par application successive de DP, un nombre exponentiel de clauses peut être ajouté dans le pire des cas. Cette méthode est connue pour être complète et permet de répondre à la question de la satisfiabilité ou à l'insatisfiabilité d'un problème SAT sous forme CNF, suivant qu'elle ait produit durant ce processus une clause vide ou pas.

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.

2.2 Énumération

L'idée de base des algorithmes énumératifs est la construction d'un arbre binaire de recherche où chaque nœud représente une sous-formule simplifiée par l'interprétation partielle courante.

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 $\neg x_i$) 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 foi.*

Définition 2.4 (arbre complet)**.** *Soit $\mathcal{F}$ une formule* CNF *et $\mathcal{V}_\mathcal{F}$ l'ensemble des variables apparaissant à $\mathcal{F}$. L'arbre binaire de recherche correspondant à $\mathcal{F}$ est dit complet si et seulement si, en plus des*

conditions exigées sur les arbres binaire 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 qu'une formule CNF $\mathcal{F}$ est satisfaite 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.*

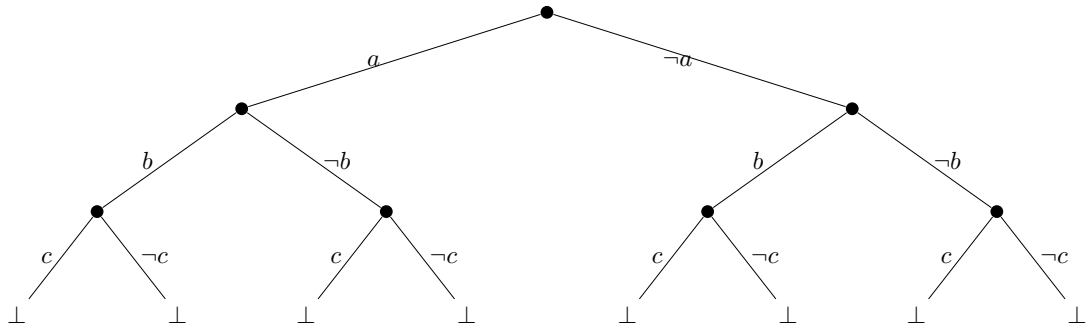


FIGURE 2.2 – Arbre binaire de recherche correspondant à l'ensemble de clauses de l'exemple 2.2.

2.3 Algorithme de Quine

L'algorithme de Quine (1950) 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, donc chaque nœud représente 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. Ça nous permet de couper les branche 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 basé sur la proposition suivante :

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

Algorithme 2.2 : QUINE

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

Début
  si $(\mathcal{F} = \emptyset)$ **alors retourner** *vrai*;
  si $(\bot \in \mathcal{F})$ **alors retourner** *faux*;
  $\ell \longleftarrow$ `HeuristiqueDeBranchement`$(\mathcal{F})$;
  retourner $($`QUINE(`$\mathcal{F}_{|\ell}$`)` *ou* `QUINE(`$\mathcal{F}_{|\neg\ell}$`)`$)$
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.*

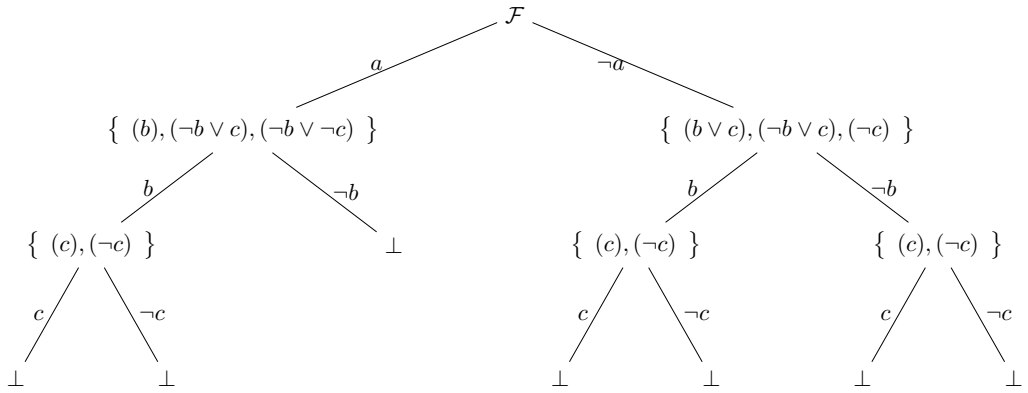


FIGURE 2.3 – Arbre construit par la méthode de Quine sur l'ensemble de clauses de l'exemple 2.2.

2.4 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, nous présentons d'abord la propagation unitaire, qui est l'un des procédés clés dans les algorithmes de résolution de SAT et qui est sans conteste la plus utilisée.

2.4.1 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.*

D'où $\mathcal{F}_{|x} = \{c | c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \setminus \{\neg x\} | c \in \mathcal{F}, \neg x \in c\}$, 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 : $p = \{x_1, ..., x_n\}$ une interprétation, 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 de clauses ne contienne plus de clauses unitaires ou jusqu'à l'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 $(\mathcal{F} \wedge \neg x) = \bot$. Une clause c est déductible par propagation unitaire de $\mathcal{F}$, si et seulement si $\mathcal{F} \wedge \bar{c} \models_* \bot$. i.e., $(\mathcal{F} \wedge \bar{c})^* = \bot$.*

Propriété 2.4. *Soit $\mathcal{F}$ 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 $(x_i) \in \mathcal{F}_{|\{x_1, x_2, ..., x_{i-1}\}}$*

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 $\mathcal{H} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$, d'où $\mathcal{S}_0$ est une séquence de propagations obtenue à partir de $\mathcal{F}$, et $\mathcal{S}_1$ est une séquence de décisions-propagations obtenue à partir de $\mathcal{F}^*$ en considérant x_1, $\forall 1 < i \leq n, \mathcal{S}_i$ 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 ($niv(\ell)$), de clause raison ($\overrightarrow{raison}(\ell)$) et d'explication ($exp(\ell)$).

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 $\ell \in \mathcal{S}_i$ alors $niv(\ell) = i$, $niv(\ell) = \infty$ 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* $\ell \in \delta$ *alors* $\overrightarrow{raison}(\ell) = \bot$*, sinon* $\overrightarrow{raison}(l) \in \mathcal{F}$ *telle que* $\ell \in \overrightarrow{raison}(\ell)$ *et pour tous les autres littéraux* $y \in \overrightarrow{raison}(\ell), \mathcal{I}(y) = \bot$ *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* $\ell \in \delta$ *alors* $exp(\ell) = \emptyset$*, sinon* $exp(\ell) = \{\neg x$ *tel que* $x \in \overrightarrow{raison}(\ell)$ *avec* $\overrightarrow{raison}(\ell)$ *une clause raison de* $\ell\}$*.*

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* $exp(g) = \{\neg e, f\}$.

2.4.2 Procédure DPLL

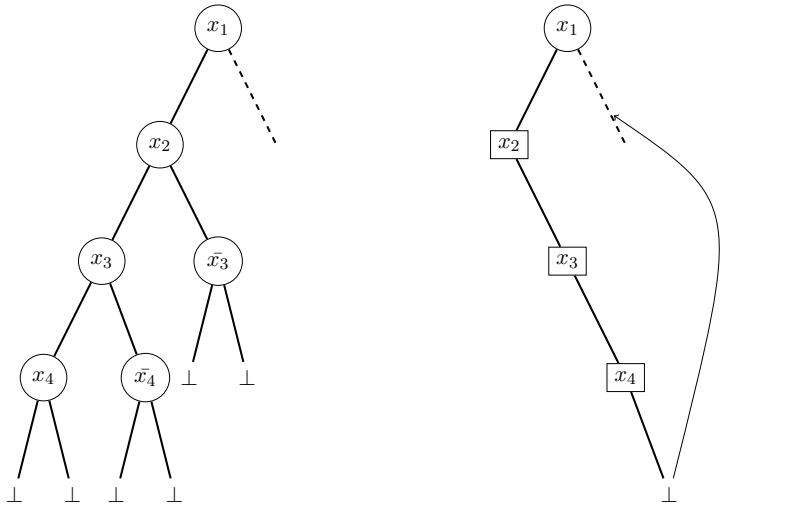
La méthode DPLL décrit par l'algorithme 2.3 est basé sur une recherche systématique dans l'espace de recherche. Elle peut être vue comme l'algorithme de Quine avec une étape supplémentaire d'inférence à chaque nœud de l'arbre : la propagation des litéraux unitaires (d'autre processus de filtrage sont possible *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. Le choix de la prochaine variable à affecter est fait suivant une heuristique(*i.e.* choix de la variable figurant le plus souvent dans les clauses les plus courtes). 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 $\mathcal{F} \wedge \ell$ et $\mathcal{F} \wedge \neg\ell$. DPLL est une méthode de type "choix+propagation".

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
```

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 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.*

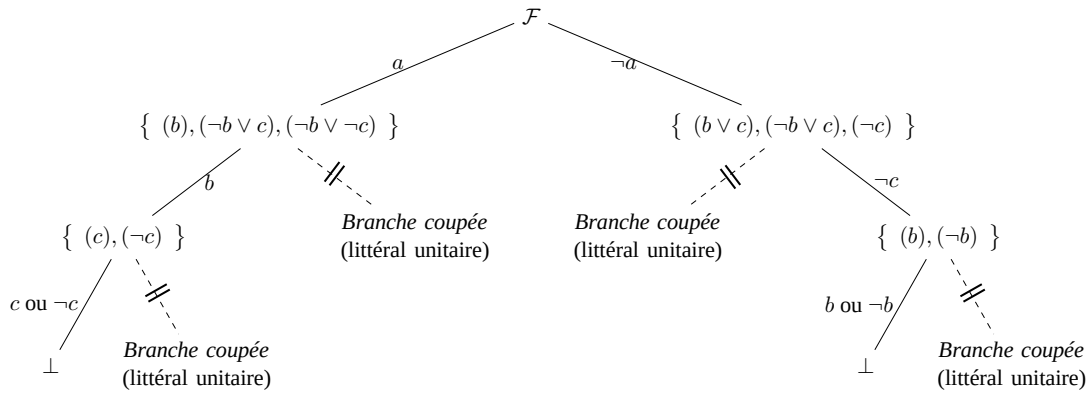


FIGURE 2.5 – Arbre de recherche construit par l'algorithme DPLL sur l'instance de l'exemple 2.2.

Par sa simplicité, la procédure DPLL est l'une des plus utilisés 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. Ce choix est fait via une heuristique. Cette heuristique de branchement a une incidence directe sur la taille de l'arbre de recherche et sera présenté plus en détail par la suite (cf 2.4.3). De nombreuses améliorations ont été apportées à la procédure DPLL, elles concernent généralement un ou plusieurs des points suivants :

1. Heuristique de branchement
2. simplification de la formule
3. traitement des échecs.

2.4.3 Heuristiques de branchements

Du fait de la relation forte entre l'ordre d'affectation des variables et la taille de l'arbre de recherche développé, une "bonne" heuristique de branchement est déterminante pour l'efficacité d'un algorithme de recherche. 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). 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, lorsque une variable est choisie comme point de choix la deuxième étape est la choix de valeur de vérité.

Heuristique de choix de variables

Nous distinguons en général trois type 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 apportés 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 permettent 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* » proposée par Goldberg (1979) 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.

Cette heuristique a été amélioré 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. 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 : $J(\ell) = \sum_{\alpha \in \mathcal{F} | \ell \in \alpha} 2^{-|\alpha|}$.

La première heuristique (JW-OS) proposée consiste à sélectionner le littéral ℓ qui maximise la fonction $J(\ell)$ 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 de type « *look-ahead* » consistent à anticiper le résultat de l'affectation d'une variable non encore affectée à l'aide de méthodes de filtrages (telles que la propagation unitaire). En d'autres termes, ce type d'approches effectue 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 $\neg\ell$ est propagé au niveau courant. De cette manière, le nombre de variables éligibles pour la prochaine variable de branchement est réduit. Ce type de méthodes fonctionnent surtout sur les problèmes là où l'apprentissage porte peu. Nous présentons ici 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 implémenté dans le solveur KCNFS est fondée sur la notion de *backbone* [1]. 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 : $\mathcal{B}(\ell)$ 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 $bin(\ell)$ (respectivement $ter(\ell)$) 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 $\mathcal{B}(\ell)$;
3 **si** $i = 0$ **alors retourner** $\sum_{(u \vee v) \in \mathcal{B}(\ell)} (2 \times bin(\neg u) + ter(\neg u)) \times (2 \times bin(\neg v) + ter(\neg v))$;
4 **retourner** $\sum_{(u \vee v) \in \mathcal{B}(\ell)} \text{BSH}(\neg u) \times \text{BSH}(\neg v)$;

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). 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 citons :

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 leurs activités 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 contrainte. 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

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

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.

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 induit sur les clauses peuvent être facilement transférés aux variables ;

Heuristiques de polarité

Bien qu'étant aussi importante que l'heuristique de choix de variables, la deuxième étape d'heuristique de branchement, l'heuristique de choix de polarité est souvent passée sous silence lors de 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 de la complexité algorithmique, ce choix est effectué pour l'instant de la mainière heuristique. Nous citons ici quelques unes des plus connues :

false/true : C'est une heuristique de phase très simple. Il consiste à toujours affecter la variable à la valeur fausse/vrai ($\neg x/x$).

JW : dans (Jeroslow et Wang (1990)), les auteurs proposent une mesure w, prenant en compte des informations sur le problème (taille et nombre de clauses contenant la variable), pour sélectionner la polarité du prochain point de choix (choisir x si $w(x) \geq w(\neg x)$, $\neg x$ sinon). La fonction permettant de donner un score à une variable proposée par les auteurs est celle décrite lors de la présentation de l'heuristique du même nom (cf. équation 2.4.3) ;

progress saving : dans Pipatsrisawat et Darwiche (2007) les auteurs observent, de manière expérimentale, que dans le cadre d'une approche utilisant le *b*acktracking les solveurs effectuaient beaucoup de travail redondant. En effet, lorsqu'un retour-arrière est effectué, le travail accompli pour résoudre les sous-problèmes traversés avant d'atteindre le conflit est perdu. Pour éviter cela, les auteurs proposent de sauvegarder la dernière polarité obtenue pendant la recherche dans une interprétation complète notée $\mathcal{I}$. Ainsi, lorsqu'une nouvelle décision sera prise, la variable se verra attribuée la même polarité que précédemment (choisir x si $x \in \mathcal{I}$, $\neg x$ sinon). De cette manière, l'effort pour satisfaire un sous-problème déjà résolu auparavant sera moins important. Le principal désavantage de cette approche est de ne pas assez diversifier la recherche. Afin de pallier ce problème, Biere (2009b) propose d'« oublier » une partie de l'interprétation $\mathcal{I}$.

occurence : l'heuristique *occurence* a été proposé par Hamadi *et al.* (2009c) et a été implémenté dans le solveur LYSAT. D'où les auteurs mesurent le poids d'un littéral ℓ par le nombre d'occurrence $\#occ(\ell)$ de celle-ci dans les clauses apprises (choisir ℓ si $\#occ(\ell) \geq \#occ(\neg\ell)$, $\neg\ell$ sinon). Cette heuristique a pour objectif de balancer le nombre de littéraux positives et littéraux négatives afin de effectuer plus de résolution.

2.4.4 Analyse de conflits et retour-arrière non chronologique

Une des caractéristiques de l'algorithme DPLL est qu'il remonte l'arbre de recherche jusqu'au nœud de décision précédent l'échec lorsqu'un conflit est atteint. L'un des problèmes majeurs du retour-arrière chronologique réside dans la non prise en compte des raisons qui ont mené au conflit. L'analyse de conflits remédie à ce problème en tenant compte des décisions à l'origine du conflit. Cette analyse permet d'effectuer un retour-arrière non chronologique, et d'éviter de redécouvrir les mêmes échecs. En effet, 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. Ce type d'approches a été étudié et appliqué dans le nombreux domaines de l'IA, comme dans les systèmes de maintien de vérité ATMS (McAllester (1980), Stallman et Sussman (1977)), les problèmes de satisfaction de contraintes CSP (Dechter (1990), Ginsberg (1993), Schiex et Verfaillie (1993)) et en programmation logique (Bruynooghe (1980)), *etc.*. Elles se distinguent essentiellement par les techniques utilisés pour analyser les échecs et pour éviter de rencontrer les mêmes situations au cours de la recherche. Dans le cadre SAT ce n'est qu'en 1996 et par l'intermédiaire de Marques-Silva et Sakallah (1996a) que l'analyse de conflit est introduit.

L'analyse du conflit permet d'identifier dans l'interprétation courante un sous-ensemble de décisions et propagations (appelé ensemble conflit) à l'origine de la dérivation de la clause vide Bayardo Jr. et Schrag (1997). Ce sous-ensemble permet de déterminer 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 2.9. *Soit $\mathcal{F}$ une formule* CNF *définie comme suite :*

$$\mathcal{F} = \begin{cases} (c_1) & (x_1 \vee x_2) \\ (c_2) & (x_2 \vee x_3) \\ (c_3) & (\neg x_1 \vee \neg x \vee y) \\ (c_4) & (\neg x_1 \vee x \vee z) \\ (c_5) & (\neg x_1 \vee \neg x \vee z) \\ (c_6) & (\neg x_1 \vee x \vee \neg z) \\ (c_7) & (\neg x_1 \vee \neg y \vee \neg z) \end{cases}$$

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

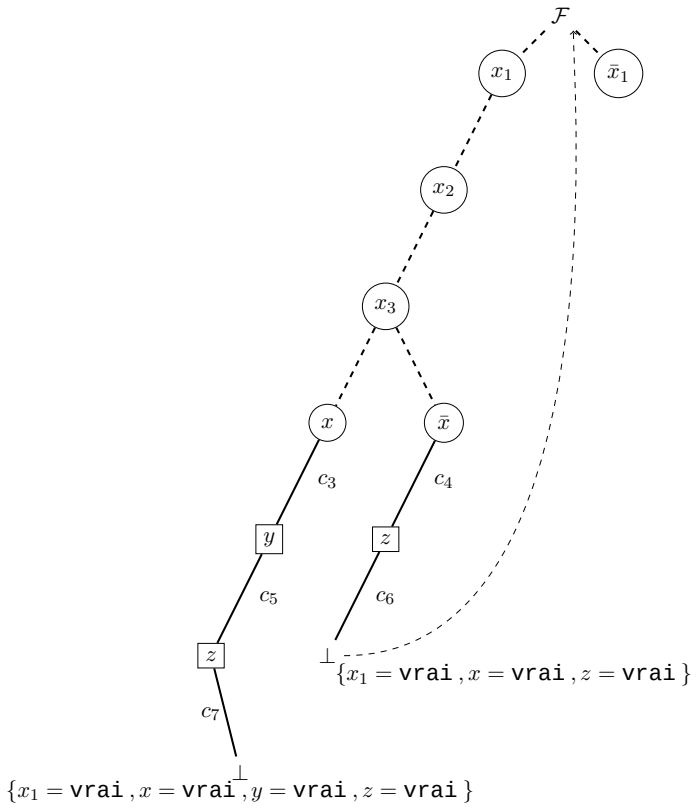


FIGURE 2.6 – Un retour-arrière non chronologique.

Après avoir affecté $x_1 = \texttt{vrai}\,, x_2 = \texttt{vrai}\,, x_3 = \texttt{vrai}\, etx = \texttt{vrai}\,$, la propagation unitaire permet de propager $y = \texttt{vrai}$ (clause c_3), $z = \texttt{vrai}$ (clause c_5) et la clause c_7 devient vide (tous les littéraux sont à $\texttt{faux}$). Dans ce cas, l'ensemble de conflit est l'ensemble des variables qui ont menés au conflit $\{x_1 = \texttt{vrai}\,, x = \texttt{vrai}\,, y = \texttt{vrai}\,, z = \texttt{vrai}\,\}$. Notons que x_2 et x_3 ne participent pas au conflit. L'algorithme teste ensuite l'autre valeur de vérité de x. Supposons qu'après avoir affecté x à $\texttt{faux}$, l'algorithme détecte un nouveau conflit et que l'ensemble de conflit est $\{x_1 = \texttt{vrai}\,, x = \texttt{vrai}\,, z = \texttt{vrai}\,\}$. Du moment qu'on a testé de manière exhaustive les valeurs de vérité de x, le retour-arrière non chronologique peut maintenant se faire au niveau de x_1 qui est affecté à $\texttt{faux}$ et le processus est réitéré. Nous pouvons remarquer que, cette fois, l'algorithme est en mesure de sortir du conflit sans essayer les décisions intermédiaires x_2 et x_3.

Le retour-arrière non chronologique permet d'éviter partiellement les mêmes conflits ou "trashing" (qui consiste à explorer de façon répétitive les mêmes sous-arbres). Il peut toutefois répéter les mêmes erreurs dans le futur. On peut remédier à ce problème en ajoutant les clauses représentant le conflit ("nogood"). Cette clause est obtenue par la négation du terme construit par analyse de conflits $(\neg x_1 \vee \neg x_2 \ldots \neg x_n)$. L'ajout de cette clause, connue sous le nom de *clause apprise* (*learnt clause*) Marques-Silva et Sakallah (1996a), Bayardo Jr. et Schrag (1997), Zhang *et al.* (2001), Beame *et al.* (2004), permet à chaque fois qu'une contradiction est détectée d'identifier et d'ajouter une clause impliquée par la formule. De plus, conserver cette clause pour la suite de la recherche permet à la propagation unitaire de découvrir de nouvelles implications lesquelles permettent d'éviter de considérer la même situation d'échec plusieurs fois.

2.5 Solveur SAT moderne

Dans la suite, nous donnons une brève description d'un solveur SAT moderne. Une description plus formelle sera donnée dans le chapitre suivant (cf chapitre 3).

Les progrès récents obtenus dans le cadre de la résolution pratique du problème SAT font généralement référence à l'efficacité remarquable des solveurs SAT (appelé solveurs SAT moderne). La mise en œuvre des solveurs SAT modernes est rendue possible (liste non exhaustive) 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.* (2003)), et *backbone* (Dequen et Dubois (2004))), à l'exploitation des propriétés structurelles(*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 dirigés par les conflits (Brisoux *et al.* (1999)) et à la proposition de structures de données efficaces (*i.e. watched literals* (Zhang (1997), Zhang *et al.* (2001))).

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

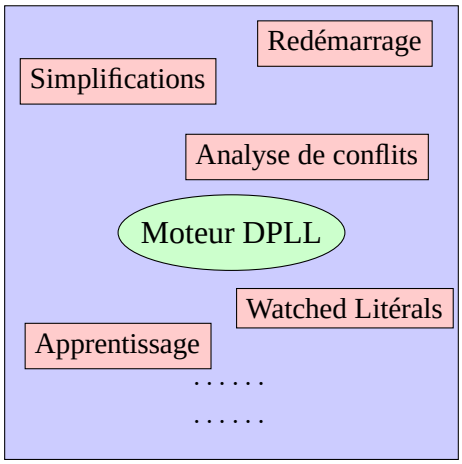


FIGURE 2.7 – Soleveur SAT moderne

C'est la combinaison de ces principes et leurs intégrations au sein de l'algorithme DPLL qui ont conduit à l'élaboration des solveurs SAT modernes (Marques-Silva et Sakallah (1996a), Zhang *et al.* (2001), Beame *et al.* (2004)).

2.6 Conclusion

Nous avons abordé dans ce chapitre la résolution pratique du problème SAT ainsi que la mise au point d'algorithmes performants pour le résoudre. Ces algorithmes de recherche reposent pour la plupart sur la procédure DPLL et sur une analyse à partir des échecs rencontrés. Cette analyse de conflits et apprentissage de *nogoods* qui en découle ont permi d'augmenter significativement la puissance des prouveurs SAT permettant aussi de résoudre des problèmes NP-complets. Nous allons voir dans la section suivante une description plus formelle des solveurs SAT modernes.

Chapitre 3

Prouveurs SAT moderne

Sommaire

DANS CE CHAPITRE, nous présentons d'abord l'architecture générale d'un solveur CDCL ainsi que ces principales composantes.

3.1 L'architecture générale

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 analyse de ce conflit (ligne 15), le plus souvent selon le schéma « First UIP » (« *Unique Implication Point* » Zhang *et al.* (2001)), permettant de déduire d'un littéral x aurait dû être affecté à un niveau plus haut dans l'arbre de recherche. Ce niveau correspond au niveau de *backjump* bl et est calculé (ligne 16). Un saut arrière arrière est alors effectué à ce niveau et le littéral x 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 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 le propagation unitaire. La propagation unitaire va ensuite conduire l'apprentissage, et l'apprentissage en revanche peut influencer l'affectation de variable.

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)

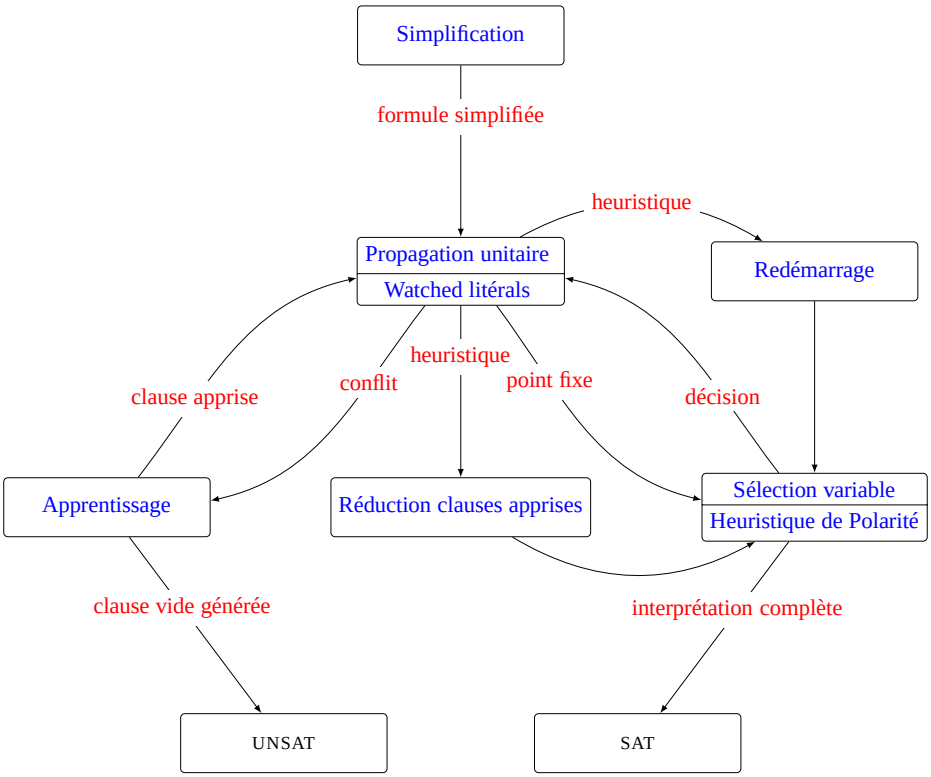


FIGURE 3.1 – Interaction des composantes d'un solveur CDCL.

Ce diagramme commence par une composante important des solveurs SAT moderne qui est le prétraitement de la formule.

3.2 Prétraitement de la formule

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.

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.

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.

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 $\ell \in \alpha$ est bloqué si $\forall \alpha' \in \mathcal{F}$ telle que $\neg\ell \in \alpha'$ la résolvante $\eta[\ell, \alpha, \alpha']$ est tautologique Heule *et al.* (2010), Järvisalo *et al.* (2010). Les clauses bloquées sont des clauses redondantes. Cependant, éliminer une clause bloquée de la formule initiale ne conserve pas 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 $\ell' \in \mathcal{L}_\mathcal{F}$ à une clause $\alpha \in \mathcal{F}$ est possible si $\exists \ell \in \alpha$ tel que $(\ell \vee \neg\ell') \in \mathcal{F} \setminus \{\alpha\}$. Cette extension est appliquée jusqu'à l'obtention d'un point fixe. Les auteurs montrent que si la clause obtenue 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* ;

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 $\mathcal{F}$, $\mathcal{I}_\ell = (\mathcal{F}_{|\ell})^*$ et $\mathcal{I}_{\neg\ell} = (\mathcal{F}_{|\neg\ell})^*$ 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 $x \in \mathcal{I}_\ell \cap \mathcal{I}_{\neg\ell}$ alors $\mathcal{F} \models (x \vee \ell)$ et $\mathcal{F} \models (x \vee \neg\ell)$ [2] 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 $\mathcal{I}_\ell$ et $\mathcal{I}_{\neg\ell}$. En effet, soit x tel que $x \in \mathcal{I}_\ell$ et $x \in \mathcal{I}_{\neg\ell}$ alors $(\ell \Rightarrow x)$ et $(\neg\ell \Rightarrow \neg x)$ et donc $(\ell \Leftrightarrow x)$;

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 $\alpha = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n)$ 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. $\mathcal{F}_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models_* \bot$ avec $i < n$. Dans ce cas la clause α peut être remplacée par la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i)$. En effet, puisque l'interprétation $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_i\}$ falsifie $\mathcal{F}$ la clause α', elle peut être ajoutée à la formule initiale. Comme α' subsume α, il est possible de remplacer α par α' ;
2. $\mathcal{F}_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models_* \ell_j$ tel que $\ell_j \in \alpha$ et $i < j < n$. Ici, le fait que $\mathcal{F}_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models_* \ell_j$ nous permet d'inférer la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i \vee \ell_j)$ laquelle subsume α. De la même manière que pour le premier cas, la clause α peut être remplacée par α' ;
3. $\mathcal{F}_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models_* \neg\ell_j$ tel que $\ell_j \in \alpha$ et $i < j \leq n$. Dans ce cas, $\mathcal{F}_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models_* \neg\ell_j$ nous permet d'inférer la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i \vee \neg\ell_j)$. Puisque, α' self-subsume α il est possible de remplacer la clause α par $\eta[\ell_j, \alpha, \alpha']$.

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.

3.3 Les structures de données paresseuses ("Watched literals")

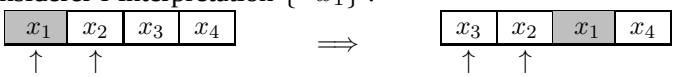
Comme nous l'avons souligné auparavant, la propagation unitaire est un processus fondamental des solveurs SAT. Dans les solveurs SAT modernes, 80% du temps de calcul est consommé par la propagation unitaire. Afin d'améliorer l'efficacité de la propagation unitaire, de nombreux progrès algorithmiques ont été effectués ces dernières années, le principal est sans doute celui proposé par Zhang et Malik (2002).

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

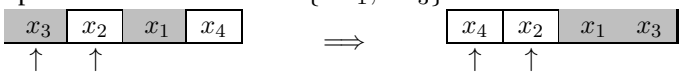
Les auteurs introduisent une structure de données dite « paresseuse » qui permet un traitement de la propagation unitaire d'une complexité moyenne sous-linéaire. En effet, la constatation est d'autant plus évidente qu'un littéral ne peut être propagé tant qu'au moins deux littéraux de cette clauses ne 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.

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 littéraux 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.

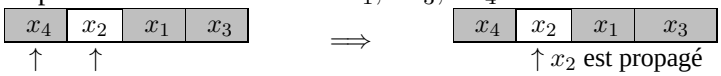
– commençons par considérer l'interprétation $\{\neg x_1\}$:



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



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



Cependant, cette structure de données paresseuse possède également quelques inconvénients : comme les clauses n'étant pas considérées dans leurs intégralité, l'utilisation d'heuristiques syntaxiques ne sont plus possibles car ils requièrent une connaissance complète de l'instance après affectation(s). On gagne en vitesse en perdant de l'information. Seules les heuristiques sémantiques peuvent être appliquées avec les structures de données paresseuses, la structure du problème n'étant pas connue.

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

L'apprentissage est la différence principale entre les procédures DPLL et CDCL. Le retour-arrière non chronologique permet d'éviter partiellement les mêmes conflits ou « trashing ». Mais il peut toutefois répéter les mêmes erreurs dans le futur. L'utilisation du retour-arrière non chronologique dans les solveurs modernes CDCL est alors couplé avec le mécanisme d'apprentissage. 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 analyse 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 représentation capturant les littéraux affectées durant la recherche, que ce soit les littéraux de décision ou des littéraux propagées. 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 ($x \in exp(y)$). Le graphe d'implications se définit formellement 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 $\langle \neg x_1, x_4, \neg x_7, x_{10}\rangle$. Le niveau de décision courant est 4. Nous représentons en rouge (respectivement vert) les littéraux de la formule falsifiées (respectivement satisfait) 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 $\mathcal{I}$.*

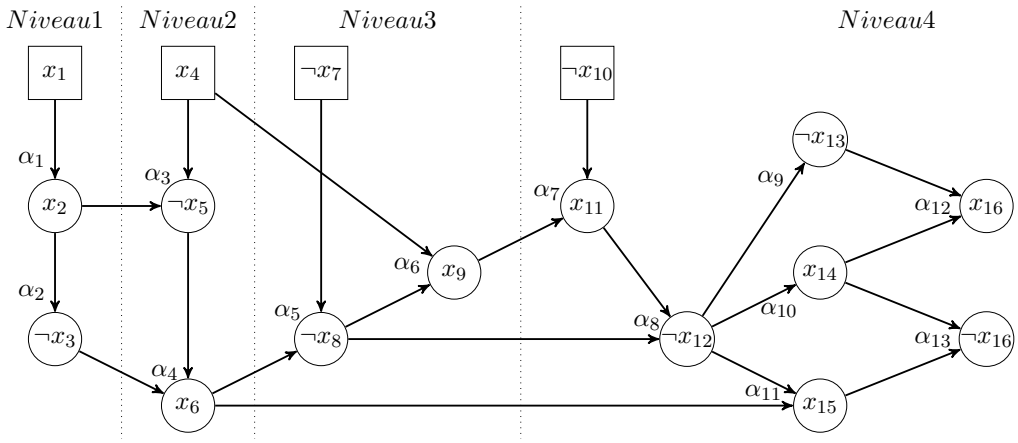


FIGURE 3.2 – Graphe d'implication obtenue à partir de la formule et l'interprétation partielle de l'exemple 3.1. Les nœuds de décision sont affichés en haut et sont représentés par des nœuds carrés. Les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'entre eux, 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, on considère $\mathcal{F}$ 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 dernier niveau de décision qui domine le conflit. Formellement, nous avons :

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 résolvantes entre les clauses responsables du conflit (utilisées durant la propagation unitaire) en remontant de celui-ci 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'extraire une clause ne possédant qu'un seul littéral du dernier niveau. Cette clause est nommée clause assertive et 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.*

$$\begin{array}{lcll}
\sigma_1 & = & \eta[x_{16}, \alpha_{12}, \alpha_{13}] & = & x_{13}^4 \vee \neg x_{14}^4 \vee \neg x_{15}^4 \\
\sigma_2 & = & \eta[x_{15}, \sigma_1, \alpha_{11}] & = & \neg x_6^2 \vee x_{12}^4 \vee x_{13}^4 \vee \neg x_{14}^4 \\
\sigma_3 & = & \eta[x_{14}, \sigma_2, \alpha_9] & = & \neg x_6^2 \vee x_{12}^4 \vee x_{13}^4 \\
\sigma_4 & = & \eta[x_{13}, \sigma_3, \alpha_{10}] & = & \neg x_6^2 \vee x_{12}^4 \\
\sigma_5 & = & \eta[x_{12}, \sigma_4, \alpha_8] & = & \neg x_6^2 \vee x_8^3 \vee \neg x_{11}^4 \\
\sigma_6 & = & \eta[x_{11}, \sigma_5, \alpha_7] & = & \neg x_6^2 \vee x_8^3 \vee x_{10}^4
\end{array}$$

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 et avoir propagé le littéral assertif est $\mathcal{I} = \{\langle(\neg x_1^1), x_2^1, \neg x_3^1\rangle, \langle(x_4^2), \neg x_5^2, x_6^2, x_{12}^2\rangle\}$.*

Plusieurs extensions du schéma d'analyse de conflits classique ont été proposées. Par exemple, dans Audemard *et al.* (2008a), les auteurs proposent d'étendre le graphe d'implications en utilisant des clauses unitaires satisfiates, généralement ignorées dans le schéma classique. Ce nouveau schéma permet d'améliorer, sous conditions, le niveau du retour arrière. D'autres approches Sörensson et Biere (2009), Han et Somenzi (2009), Hamadi *et al.* (2009b) et Nadel et Ryvchin (2010) ont été proposées permettant d'obtenir d'autres extensions.

Les clauses générées à partir du schéma d'analyse de conflits sont stockés dans une base appelée base de clauses apprises. Cependant le nombre des clauses générées est connu pour être exponentiel. Dès lors la mémoire et la propagation unitaire se retrouvent affectés 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 la base de clauses apprises 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 scores et les clauses considérées peu importantes seraient enlevées. 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. Nous présentons dans le section suivante des critères permettant d'évaluer la qualité des clauses apprises et des stratégies de nettoyage de la base des clauses apprises.

3.5 Heuristiques de réduction de la base de clauses apprises

L'analyse de conflits et l'apprentissage ont permis d'améliorer de manière significative l'algorithme DPLL Marques-Silva et Sakallah (1996a). Cependant, comme le font justement remarquer, 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 $n/2$

```
si faireReduction() alors
    TrierLesClausesApprises;        /* Selon les critères définis */
    limit = n/2;
    ind = 0;
    tant que ind < limit faire
        clause = Δ[ind] ;
        si clause.size() > 2 alors
            EliminerLaCLause;
        sinon
            GarderLaClause;
retourner Δ ;
```

L'algorithme 7.1 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 Les 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 systématiquement 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 ce qui est 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 basée 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 détaillons ces diverses procédures ci-dessous.

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, et consiste à augmenter le poids des clauses participant au 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 âges. 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 basée 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 initialisée 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 basée 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. Partant de ce constat, ils proposent d'utiliser ce critère afin d'estimer l'importance de clause apprise. Les valeurs de PSM pour chaque clause apprise est calculé et ajusté à un certain fréquence.

3.5.2 La fréquence de la réduction(Quand ?)

Un autre point important concerne la fréquence avec laquelle la base de clauses apprises est nettoyée. Cette fréquence, calculée de manière heuristique (fonction `faireReduction`), et si elle est mal gérée, peut conduire à rendre le solveur incomplet. En effet, contrairement à 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 supprimer une clause peut amener le solveur à parcourir plusieurs fois le même espace de recherche et donc à visiter encore et encore le même conflit sans jamais s'arrêter. 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 cette intervalle permet d'assurer la terminaison de l'algorithme quelque soient les clauses supprimées précédemment. À l'heure actuelle deux approches sont principalement utilisées afin de gérer cette intervalle. La première approche, implémentée dans MINISAT Eén et Sörenson (2004), consiste à appeler la fonction de réduction de la base de clauses apprises une fois que la taille de celle-ci a atteint un certain seuil. Ce seuil initialisé à $1/3$ de la taille de la base de clauses initiale est augmentée à chaque $100, 150, \ldots, 100 * 1.5^n$ conflits de 10%. La deuxième approche est la stratégie implantée dans le solveur GLUCOSE Audemard et Simon (2009a). Elle 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 Le nombre de clauses à conserver/enlever (Combien ?)

Dans la plupart de solveurs SAT de l'état de l'art, lorsque la fonction de réduction est appelée, l'ensemble des clauses apprises sont triées selon les différents critères, et un certain seuil est calculé à partir des critères utilisés (par exemple dans MINISAT, un seuil calculé par le compteur d'augmentation et la taille de la base est définie), et la moitié des clauses estimées comme moins importantes plus les clauses qui ont des valeurs en dessous du seuil sont supprimées.

En effet, il n'y pas de justification ou preuve qui nous permet de définir le nombre optimal de clauses à garder. Dés lors décider du nombre exacte de clauses à garder reste une question ouverte.

Dans le chapitre 7, nous allons montrer un point de vue global sur les trois questions importantes : (Lesquelles, Combien et Quand ?) et aussi les comportements des différentes combinaisons des ces trois parties.

3.6 Heuristiques de Redémarrage

Les premiers choix effectués lors d'une recherche complète sont prépondérants. En effet Gomes *et al.* (2000) ont montré expérimentalement qu'exécuter la même approche sur le même problème mais avec des ordonnancements de variables différents conduit à des temps de résolution totalement hétérogènes. 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).

Ce constat a été la base de l'introduction des redémarrages dans les solveurs SAT modernes. il consiste à démarrer une nouvelle recherche à priori avec un nouvel ordre de variables. La plupart des solveurs effectuent un redémarrage après qu'un certain seuil de clause apprise (ou de conflits) est atteint. 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.

Une autre observation faite dans Ryvchin et Strichman (2008) montre 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.

Dans la suite nous présentons les stratégies de redémarrages les plus souvent utilisées (fonction `redémarrage`). Ces stratégies se décomposent en deux catégories. La première classe regroupe les stratégies de redémarrages statiques. La seconde classe regroupe les approches dynamiques visant à exploiter différentes informations en cours de recherche afin de déterminer le moment d'effectuer un redémarrage.

3.6.1 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. Voici quelques stratégies de redémarrages utilisées par différents solveurs :

Intervalle fixe : Cette première stratégie somme toute très naïve consiste à appeler la fonction de redémarrage tous les x conflits. Elle est utilisée dans différents solveurs tels que SIEGE Ryan (2004), ZCHAFF Moskewicz *et al.* (2001) (version 2004), BERKMIN Goldberg et Novikov (2002) et EUREKA Nadel *et al.* (2006) avec respectivement $x = 16000$, $x = 700$, $x = 550$ 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). Ce solveur commence par initialiser le premier intervalle de conflit autorisé à $limite = 100$. Ensuite, lorsque le nombre de conflit atteint la valeur $limite$, un redémarrage est effectuée et la valeur limite est augmentée de 50% ($limite = limite \times 1.5$) ;

Luby : La série de Luby a été introduite dans Luby *et al.* (1993) et a été utilisée pour la première 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 caractéristiques 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 (2009b) qui l'utilise avec $u = 100$;

La figure 3.4 montre l'évaluation de la série de luby avec $u = 512$.

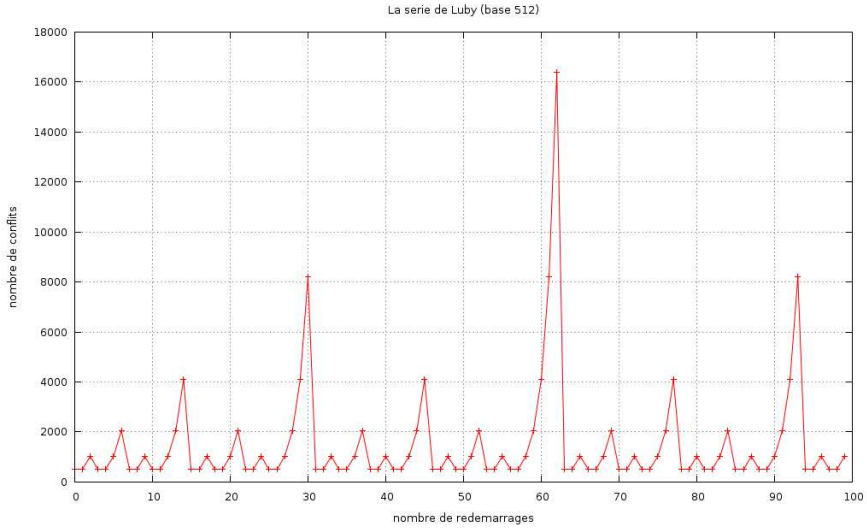


FIGURE 3.3 – La série de luby avec $u = 512$

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.

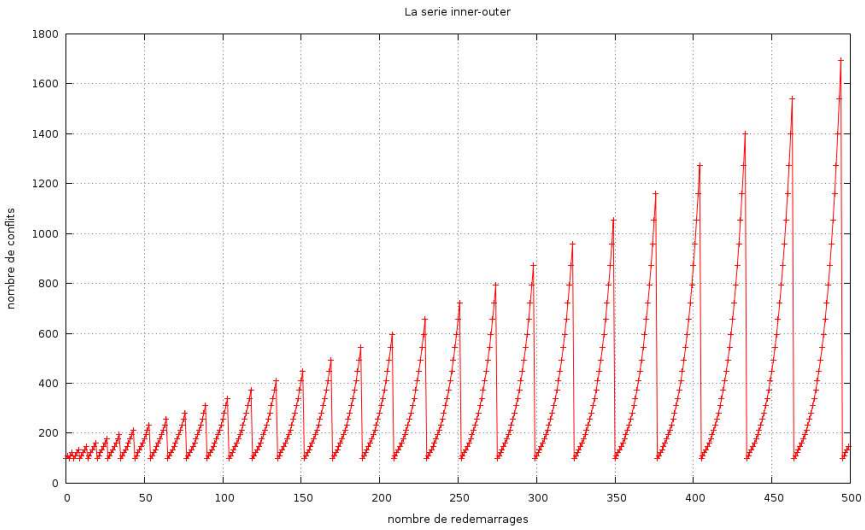


FIGURE 3.4 – La série géométrique inner-outer avec $inner = \{100, 110, \ldots, 100 * 1.1^n\}$,$outer = \{100, 110, \ldots, 100 * 1.1^n\}$

L'impact de toutes ces différentes stratégies de redémarrages sur le comportement d'un solveur SAT moderne ont été étudiée dans Huang (2007).

3.6.2 Stratégies de redémarrage dynamiques

Depuis quelques temps, des stratégies de redémarrages dynamiques sont proposées. Elles tentent d'exploiter différentes informations issues de la recherche (la taille des résolvantes Pipatsrisawat et Darwiche (2009), la profondeur moyenne de l'arbre et des sauts arrières Hamadi *et al.* (2009c), le nombre récent de changements de valeurs de variables Biere (2008a)) afin de déterminer quand effectuer un redémarrage. Nous présentons dans la suite une liste non exhaustive de ces différentes approches :

Variation de la phase : Cette approche proposée par Biere (2008a) est l'une des premières politiques de redémarrages dynamique. L'idée est d'étudier l'agilité de la recherche afin de décider si un redémarrage est nécessaire. Cette agilité est calculée par l'analyse de l'interprétation complète représentant le *progress saving* (voir §2.4.3). 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 [3] 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[3]

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

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 a été proposée par Hamadi *et al.* (2009c) et a été implémentée dans le solveur LYSAT. Elle est basée sur l'évolution de la taille moyenne des retours arrière(le moyenne des hauteur des sauts de chaque retour arrière). L'idée est de délivrer pour de grandes (respectivement petites) fluctuations de la taille moyenne des retours arrière (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 arrière 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 conflit, 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 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 ;

3.7 Conclusion

Dans ce chapitre, nous avons présenté les solveur SAT moderne et ses différentes composantes. Depuis la naissance des solveurs SAT modernes, des améliorations sur ces composants essentiels ont eu un impact significatif sur sa performance. Chaque composant a de l'influence directe sur la performance du solveur et aussi sur les interactions entre les différents composants. Ces dernières années, des progrès importants ont été réalisé dans les solveurs SAT moderne. Cependant, les dernières évaluations et compétitions internationales (SAT Race et SAT competition) montrent qu'une catégorie d'instances restent toujours difficiles pour les solveurs actuels. Dans ce contexte, avec la vulgarisation des architectures multi-cœurs, les approches parallèles offrent une vraie perspective pour s'attaquer à ces instances.

Chapitre 4

Résolution parallèle du problème SAT

Sommaire

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. Cependant, 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 dernières années, grâce aux nouvelles architectures multi-cœurs, les approches de résolution parallèle du problème SAT gagnent en intérêt.

4.1 Motivation

La loi de Moore Moore (1998) a été exprimée dans « Electronics Magazine » par Gordon Moore, un des trois fondateurs d'Intel. Constatant que la complexité des semi-conducteurs proposés en entrée de gamme doublait tous les ans. En 1975, Moore réévalua sa prédiction en prédisant que le nombre de transistors des microprocesseurs sur une puce de silicium doublera tous les deux ans. Dérivée de ces affirmations, la célèbre loi de Moore s'énonce maintenant ainsi :

Définition 4.1 (Loi de Moore)**.** *Le nombre de transistors au sein d'un processeur doublera tous les deux ans.*

Bien qu'il ne s'agit pas d'une loi physique mais juste d'une extrapolation empirique, cette prédiction s'est révélée jusqu'à présent étonnamment exacte. Entre 1971 et 2001, la densité des transistors a doublé

chaque 1.96 année. La figure 4.1 [4] montre la croissance du nombre de transistors dans les microprocesseurs Intel par rapport à la loi de Moore, en vert, la fausse hypothèse (qui est le plus souvent diffusée) voulant que ce nombre double tous les 18 mois.

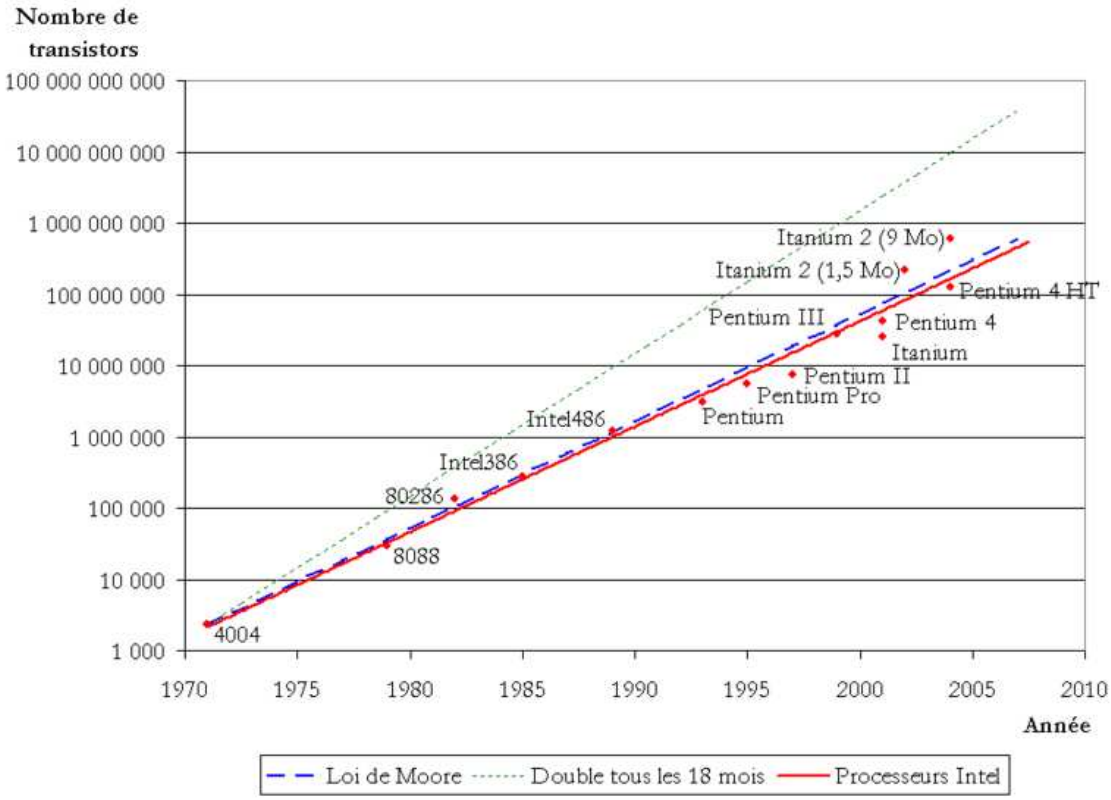


FIGURE 4.1 – Loi de Moore et réalité

En générale, on peut dire que la puissance des microprocesseurs double tous les deux ans. Entre le début de la micro-informatique et peu après les années 2000, ce terme de puissance peut être traduit par nombre de transistors mais aussi par des fréquences de fonctionnement des microprocesseurs. Cela concerne le nombre d'instruction traitées dans un laps de temps donné. Par conséquent, sans effort de programmation supplémentaire, il était possible d'accélérer un programme séquentiel en changeant simplement le processeur qui l'exécute. Depuis environ 2005, et à cause de l'échauffement trop important des microprocesseurs à des fréquences approchant les 4GHz, les fabriquants de microprocesseurs traditionnels ont des difficultés à proposer des microprocesseurs au silicium au-delà de cette vitesse de fonctionnement. Mais la loi de Moore est toujours vraie, il est toujours possible d'augmenter le nombre de cœurs de calcul au sein d'un même microprocesseur. Ce sont les processeurs multi-cœurs.

Définition 4.2 (architecture multi-cœurs)**.** *Une architecture multi-cœurs peut-être vue comme un ensemble de processeurs ayant la possibilité de communiquer ensemble. Chaque processeur peut être vu comme une unité de calcul permettant d'exécuter une seule tâche à la fois.*

Les programmes séquentiels ne sont donc plus accéléré par une augmentation de la fréquence. En effet, les solveurs SAT séquentiels sont issues du modèle de programmation séquentiel. Ces programmes

4. Image issue de la page Wikipédia française concernant la loi de Moore. Sous licence *Créative Commons paternité partage à l'identique*

sont désignés pour n'être exécutés que par un seul cœur de calcul. Afin de pleinement profiter des nouvelles avancées de l'architecture multi-cœurs, il est nécessaire d'évaluer les solveurs SAT, il faut penser parallèle.

4.2 Solveurs SAT en parallèle

Avant de présenter les solveurs SAT parallèles modernes, nous présentons tout d'abord quelques concepts de base nécessaires à la compréhension des différents solveurs parallèles mais également quelques notions pour évaluer les performances des solveurs SAT parallèles. En ce qui concerne les aspects technologiques liés au parallélisme, nous choisissons de ne pas les présenter. Néanmoins, la thèse de Vander-Swalmen (2009) peut être consulté pour de plus amples détails.

Définition 4.3 (parallélisme)**.** *Le parallélisme en Informatique donne la possibilité d'exécuter plusieurs instructions simultanément.*

Définition 4.4 (unité de calcul, exécution)**.** *Une seule unité de calcul peut exécuter une seule tâche, c'est une exécution séquentielle.* n *unités de calculs peuvent exécuter* n *tâches distinctes simultanément : c'est une exécution parallèle.*

Définition 4.5 (temps d'exécution)**.** *Le temps d'exécution est le temps écoulé entre le lancement d'un programme et l'obtention du résultat. Pour différents types de programmes, on a :*

- $\mathcal{T}_{seq}$ *: le temps d'exécution d'un programme séquentiel.*
- $\mathcal{T}_{//(n)}$ *: le temps d'exécution en parallèle sur* n *processeurs, c'est à dire l'instant où le dernier processeur a terminé son travail (*$\mathcal{T}_{seq}$ *est le cas particulier où* $n = 1$*).*

Définition 4.6 (accélération)**.** *L'accélération, notée* $\mathcal{A}cc(n)$ *sur* n *processeurs, représente le nombre de fois que le programme a été accéléré par son exécution parallèle et est donnée par la formule suivante :* $\mathcal{A}cc(n) = \frac{\mathcal{T}_{seq}}{\mathcal{T}_{//(n)}}$.

Définition 4.7 (efficacité)**.** *L'efficacité d'un programme parallèle, noté* $\mathcal{E}ff(n)$ *pour une exécution sur* n *processeurs, est définie par* $\mathcal{E}ff(n) = \frac{\mathcal{A}cc(n)}{n}$*. Ce nombre est l'accélération pondérée par le nombre de processeurs utilisés. Il représente donc la qualité de la parallélisation en fonction du nombre de processeurs.*

Bien que le problème SAT n'est pas facilement parallélisable (car une caractéristique importante de ce problème est qu'il est très dur de prédire à l'avance le temps nécessaire à l'exploitation d'une espace de recherche), deux types d'approche pour résoudre SAT en parallèle sont principalement utilisés à l'heure actuelle.

Le premier est basé sur le principe « *diviser pour régner* ». Il consiste à partager le travail entre l'ensemble des processus. Le second est de type *portfolio*, et vise à mettre en compétition plusieurs approches sur le même problème. Ce dernier est très facile à mettre en place et est particulièrement bien adapté aux approches paramétrables. Dans la suite, nous présentons sommairement l'application de ces deux modèles pour la résolution parallèle de SAT.

4.2.1 Approche 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 4.2 et 4.3 montrent la différence entre ces deux types de division. Dans 4.3.3, on trouve une présentation détaillée.

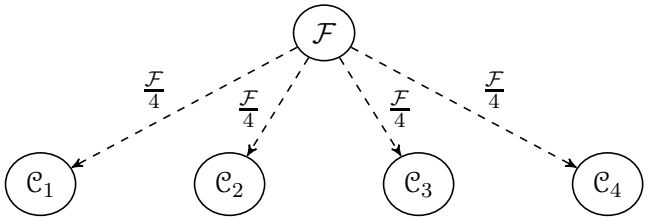


FIGURE 4.2 – Présentation schématique de la division de l'instance. Les cœurs représentent les solveurs séquentiels. Ils prennent en charge chacun une partie de la formule (ici un quart).

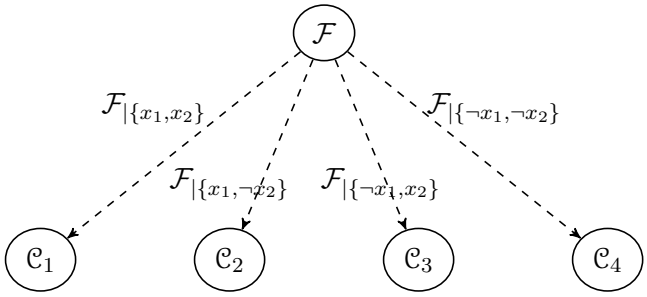


FIGURE 4.3 – Présentation schématique de la division de l'espace de recherche. Chaque cœur représente un solveur séquentiel. Ils prennent en charge chacun la formule avec un chemin de guidage.

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

Définition 4.8 (chemin de guidage)**.** *Un chemin de guidage est représenté par un ensemble de couples* $\{\langle \ell_1, \lambda_1\rangle, \langle \ell_2, \lambda_2\rangle, \ldots \langle \ell_n, \lambda_n\rangle\}$*, où* ℓ_i *est un littéral à propager et* λ_i *indique si l'un des sous-arbres issu de* ℓ_i *n'est ni en cours de traitement ni traité. Chaque* λ_i *peut être représenté par une variable booléenne qui est égale à* $\bot$ *si les deux sous-arbres sont traités (ou en cours de traitement) et* $\top$ *sinon.*

Une représentation du chemin de guidage est donné dans la figure 4.4.

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 est lui est assigné par le biais d'un des modèles d'équilibrage de charge présenté précédemment. Lorsque l'ensemble des chemins de guidage a été exploré le solveur peut conclure sur la satisfiabilité de la formule.

Nous donnons ici une liste non exhaustive de solveurs de type *diviser pour régner* 4.1.

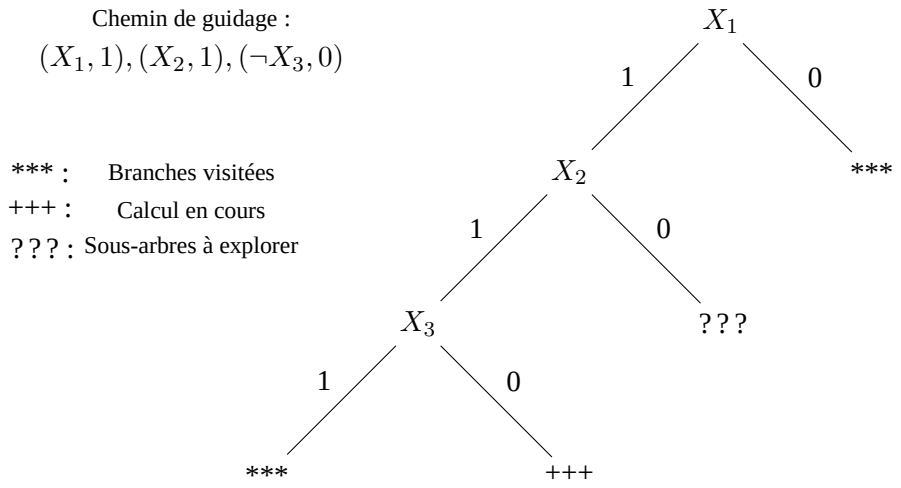


FIGURE 4.4 – Un exemple de chemin de guidage

Solveur	Algorithme de base	Architecture	Heuristique parallèle
PSato	Sato	workstation	équilibrage de charges
//Satz	Satz	workstation	équilibrage de charges
PaSat	zchaff	workstation	équilibrage de charges, échange de clauses
GradSat	zchaff	workstation	équilibrage de charges, échange de clauses
MiraXT	minisat	multicœur	équilibrage de charges, échange de clauses
PMinisat	minisat	multicœur	équilibrage de charges, échange de clauses
MTSS	minisat	multicœur	équilibrage de charges, échange de clauses

TABLE 4.1 – Divisé pour régner dans SAT

PSATO Introduit par Zhang *et al.* (1996), il est le premier solveur parallèle a introduire la notion de chemin de guidage. Il est basé sur le solveur séquentiel SATO (*Satisfiability Testing Optimized*) Zhang (1997) et utilise un modèle maître-esclaves pour partager les tâches. Le processeur maître est chargé de fournir aux processeurs esclaves les sous-arbres qu'ils doivent explorer. Afin de pouvoir toujours alimenter les processeurs esclaves, le processeur maître arrête parfois son exécution de manière à créer de nouveaux chemins de guidage. Pour cela, un ou plusieurs processeurs esclaves sont stoppés pour leur demander de retourner leur chemin courant. À partir de ces chemins et de la même manière que celle présentée précédemment, plusieurs nouveaux chemins sont créés.

//SATZ Ce solveur Jurkowiak *et al.* (2001; 2005) est un parallélisation du solveur SATZ Li et Anbulagan (1997). Conçu au tour d'une architecture de type maître-esclaves, $//Satz$ utilise un équilibrage de charges dynamique par vol de travail(*work stealing*) et à l'initiative du demandeur. Cette parallélisation permet d'évaluer la charge de travail de chaque processus esclave afin de mieux répartir les charges de travail. Grâce à la règle de branchement puissante du SATZ, aucun phénomène de ping-pong n'a été observé dans $//Satz$.

PASAT Ce solveur est le premier à permettre l'échange de clauses en plus du traditionnel chemin de guidage Sinz *et al.* (2001), Blochinger *et al.* (2003). Par la force des choses, il a aussi été le premier

confronté aux problèmes liés à ce type d'échanges, à savoir le nombre et la taille des clauses à échanger. En effet, les clauses pouvant être apprises étant nombreuses (exponentiel dans le nombre de variables de la formule) et souvent de grande taille, il est nécessaire d'appliquer une politique de limitation des clauses échangées. Le critère choisi afin de réaliser cela est fonction de la taille de la clause.

Par la suite, plusieurs approches basées sur PASAT ont été proposées. Par exemple, Chrabakh et Wolski (2003) proposent une approche nommée GRADSAT conçue pour être exécutée sur une grille de calcul. GRADSAT utilise un modèle maître/esclave où le maître distribue le travail parmi les esclaves et où chaque esclave correspond à un solveur ZCHAFF Moskewicz *et al.* (2001). Les clauses apprises par les esclaves sont gérées par le maître à l'aide d'une base de données distribuée de telle manière que seules les clauses d'une taille inférieure à une certaine constante sont partagées.

MIRAXT Ce solveur Lewis *et al.* (2007) est un re-implémentation amélioré du solveur MIRA Lewis *et al.* (2004) qui est capable de lancé sur l'architecture multicœurs. Il ne contient pas de maître comme les autres solveur parallèles. Il utilise un *Master Control Object(MCO)* qui control les communications entre les threads. Dans MIRAXT, les clauses sont en lecture seule, il utilise un autre structure paresseuse nommé *Watched Literals Reference List(WLRL)* pour la mise à jour de la formule pendant la résolution.

PMINISAT Ce solveur, introduit par Chu et Stuckey (2008), est une parallélisation simple de MINISAT 2.0 Sörensson et Eén (2009) conçu pour être exécuté sur une machine à mémoire partagée. Les chemins de guidage sont conservés au sein d'une base commune, elle même alimentée par des chemins de guidage générés par la plus longue exécution dans l'arbre. Les processus récupèrent un chemin de guidage depuis cette base lorsqu'ils en ont besoin. La particularité de ce solveur est qu'il exploite les connaissances sur les chemins des processus pour améliorer la qualité des clauses échangées. L'idée provient du fait que certaines clauses ne sont utiles que localement dans un sous-arbre. En effet, bien que les clauses puissent être de grande taille, elles peuvent devenir plus petites et par conséquent plus intéressantes après un certain nombre d'affectations. Ce principe permet à PMINISAT d'étendre le partage de clauses du moment que celles-ci deviennent petites dans un certain contexte de recherche.

MTSS Dans ce solveur *Multi-Threaded SAT Solver(MTSS)* Vander-Swalmen (2009), les auteurs ont proposé des algorithmes pleinement collaboratifs travaillent ensemble sur un même objet de grande taille avec l'arbre de guidage qui est un objet totalement parallèle, mais où le travail par nœud est séquentiel. Cet objet est mis à jour par deux types de processus : le riche qui garantie la complétude et la terminaison de l'algorithme et les pauvres qui aident à la résolution du problème. Les nouveautés apportées par ce solveur , sont les différenciations de ces deux types de processus et leur collaborations pour explorer l'arbre de recherche qui crée l'arbre de guidage, ainsi que la possibilité d'ajouter des tâches spécifiques aux processus pauvres. Le solveur effectue éventuellement en échange de rôle entre les processus riches et pauvres.

4.2.2 Approche de type portfolio

Les approches de type portfolio consistent à mettre en concurrence plusieurs solveurs séquentiels ayant chacun sa propre stratégie de recherche. 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.

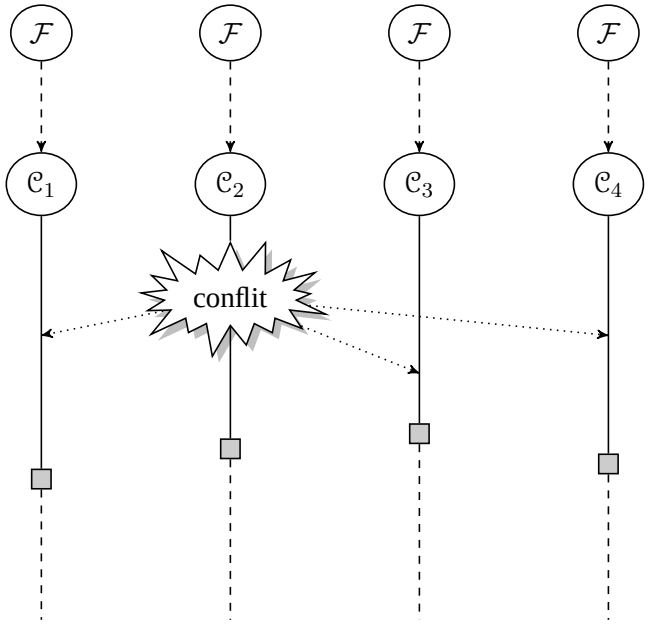


FIGURE 4.5 – Présentation schématique du solveur parallèle de type portfolio. Chaque cœur représente un solveur séquentiel (les stratégies de recherche sont différenciées). Ils prennent en charge chacun la formule originale. Les lignes continues représentent l'exécution d'un solveur. Les carrés symbolisent les redémarrages tandis que les flèches en pointillés représentent le partage d'informations entre les différents solveurs (ici les clauses apprises).

Nous donnons ici une liste non exhaustive de solveurs de type *portfolio*.

MANYSAT MANYSAT Hamadi *et al.* (2009d) a été le premier solveur parallèle de type portfolio, il inclue toutes les composantes importantes des solveurs SAT modernes (propagation unitaire, *w*atched litterals, VSIDS, analyse de conflits, redémarrage, etc.). Ce solveur tente de tirer avantage de la sensibilité aux réglages des paramètres des solveurs CDCL. En effet, changer la politique de redémarrages ou l'heuristique de choix de variables peut conduire à dégrader ou à améliorer les performances d'un solveur. Partant de ce constat les auteurs proposent d'exécuter plusieurs versions du solveur MINISAT 2.02 Eén et Sörenson (2004) avec des paramètres différents tout en partageant entre toutes les unités de calculs les clauses apprises. La première place obtenu par MANYSAT dans la catégorie parallèle de la SAT *Race* de 2008 et à la compétition SAT de 2009 a été un facteur important dans le développement des approches de type portfolio.

PLINGELING Dans Biere (2010), l'instance originale est dupliquée par un cœur maître et est distribué aux esclaves. Les stratégies utilisées par les esclaves (prétraitement, graines aléatoires, heuristique de branchement, etc) sont différenciés. L'échange de clauses apprises est fait par le maître et est restreint aux clauses unitaires.

SArTagnan Dans Kottler (2010a), plusieurs solveurs séquentiels sont exécutés en parallèle. Ces solveurs sont différenciés par rapport à heuristique de redémarrage et les paramètres de VSIDS. Une partie des solveurs appliquent le processus de résolution dynamique Biere (2009a) ou exploitent les points de référence Kottler (2010b). Les autre solveurs essaient de simplifier la base des clauses apprises en remplaçant ou éliminant des variables.

Antom Dans T.Schubert et B.Becker (2010), les solveurs séquentiels sont différenciés sur l'heuristique de décision, l'heuristique de redémarrage et la politique de réduction de la base des clause apprises, *etc.*.

Afin d'obtenir une approche portfolio efficace, il est important que les différents solveurs mis en concurrence adoptent des stratégies de recherche différenciés et complémentaires. Pour faire cela, les auteurs ont différenciés leur solveurs sur les stratégies de redémarrage, de choix de variables, de choix de polarité, et d'apprentissage (notre proposition de diversifier les stratégies de nettoyage de la base de clauses apprise sera présenté en chapitre 7).

Nous détaillons ici dans tableau 4.2 les heuristiques utilisé dans le solveur MANYSAT 1.1.

Stratégie	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$
Redémarrage	$dynamic^+$	$dynamic^-$	$geom(100, 1.5)$	$dynamic^+$
Heuristique	VSIDS (98%)	VSIDS (98%)	VSIDS (98%)	VSIDS (97%)
Polarité	*progress saving*	*progress saving*	*false*	*occurence*
Apprentissage	first-UIP	first-UIP	first-UIP étendu	first-UIP
Partage de clauses	taille ≤ 8	taille ≤ 8	taille ≤ 8	taille ≤ 8

TABLE 4.2 – Stratégies des différents solveurs séquentiels utilisés dans le solveur MANYSAT 1.1.

4.3 Défis du problème SAT en parallèle

Dans la compétition SAT 2011, un solveur (ou un shell) s'appelle ppfolio Roussel (2011) a relancé la course en résolution parallèle du problème SAT en gagnant plusieurs médailles. L'auteur propose une approche naïve qui consiste à choisir 5 solveurs séquentiels et les exécuter en parallèle sans possibilité d'échange.

Il y a donc encore un long long chemin à parcourir. Les solveurs parallèle, à priori, sont loin de maturité.

Dans Y.Hamadi et C.M.Winter (2012), les auteurs ont présenté 7 défis de la résolution parallèle du problème SAT, nous citons ici 5 qui nous semblent les plus intéressants.

4.3.1 CDCL + Recherche Locale ?

Comme nous avons présenté précédemment, les solveurs parallèles ont leur algorithmes de base, la plupart sont actuellement basés sur les solveurs CDCL. Dans les solveurs CDCL, environ 80% du temps est passé à effectuer la propagation unitaire, qui est difficile à paralléliser. L'accélération de ces solveurs parallèles par rapport au solveurs séquentiels est principalement obtenu par la décomposition de l'espace de recherche et la diversification en combinant les différentes heuristiques des solveurs séquentiels.

Rappelons la victoire du solveur ppfolio dans la dernière compétition, une combinaison entre les algorithmes complets et les algorithmes incomplet semble intéressante.

4.3.2 Équilibrage de charge

Afin d'élaborer un algorithme parallèle efficace, il y a une question primordiale à considérer, c'est la distribution de la charge entre les processeurs, plus particulièrement l'équilibrage de charge à la volé.

Définition 4.9 (partage de charge)**.** *Le partage de charge est la technique pour distribuer le travail aux unités de calcul disponibles.*

Le partage de charge idéal est d'équilibrer la taille des sous-arbres explorés par chacun des processeurs. Cela permet des appels récurrents au maître afin d'obtenir de nouveaux chemins de guidage et par conséquents de diminuer le temps consacré au partage des tâches. Mais, SAT est un problème dont la résolution à l'étape i est dépendante des calculs à l'étape $i-1$, la parallélisation est plus difficile, de plus, si on prend la représentation en arbre binaire de la procédure DPLL, il est impossible de prédire exactement le temps que prendra le calcul d'un sous-arbre. Ces points obligent les chercheurs de la communauté SAT à prendre en compte la problématique de l'équilibrage de charge.

Définition 4.10 (équilibrage de charge)**.** *L'équilibrage de charge est la technique de distribuer du travail à toutes les unités de calculs disponibles, et ce, de manière équitable afin de minimiser le temps où les unités de calculs sont oisives.*

Une piste possible est d'étendre les techniques de réglage automatique. Ces approches Xu *et al.* (2008) utilisent des techniques « d'apprentissage » pour élaborer une fonction prédictive qui estime le temps de calcul en fonction des caractéristiques d'une instance et des paramètres du solveur. Cette fonction peut être entraînée et testée sur un grand nombre d'instances représentatives et puis utilisé à configurer le solveur. L'inconvénient de cette approche est qu'il faut disposer d'un grand nombre d'instances représentatives.

4.3.3 Décomposition

Pour les algorithmes parallèles, c'est naturel de penser à décomposer le problème original en un ensemble de sous-problèmes, la plupart des solveurs SAT parallèles sont basés sur les algorithmes de recherche, dont on peut les identifier en deux types de décomposition :

- Décomposition de l'espace de recherche ;
- Décomposition de l'instance.

Dans la première catégorie, comme nous l'avons présenté, le but est de distribuer les chemins de guidages qui représentent des sous-espace de recherche. À partir de l'expérience passée, trouver une bonne décomposition de ce type est très difficile à cause de la difficulté de prédire la dureté de chaque sous-problème.

Dans la seconde catégorie, l'instance originale est divisée « en aveugle » tel qu'aucune information supplémentaire sur le problème n'est connu. Ce type de décomposition est très important et nécessaire quand l'instance originale est très grande. Trouver une décomposition optimale qui équilibre les tailles des sous-problèmes n'est pas difficile pour le problème SAT, cependant se restreindre au seul critère de la taille, n'est pas suffisant pour avoir des sous-problèmes à difficultés similaires. Et encore, trouver une décomposition d'une instance qui minimise le nombre de variables partagés est une question difficile.

Bien évidemment, pour ces deux types de décomposition, l'état de l'art n'est pas suffisant, des algorithmes de décomposition plus performants sont nécessaires pour les deux catégories.

4.3.4 Pré-traitement

Dans les années passées, plusieurs techniques de pré-traitement des formules SAT ont été proposées (*i.e.* Eén et Biere (2005)). Ces approches ont considérablement amélioré les performances des solveurs séquentiels. Nous pensons que de nouvelles techniques de pré-traitement pour la résolution parallèle du problème SAT sont indispensables. Par exemple, on a probablement éliminé trop de clauses avant la décomposition, en plus, le pré-traitement dans le contexte du parallèle devrait prendre en compte la spécificité des approches parallèles, surtout le type de décomposition utilisée. Par exemple, pour la décomposition d'une instance, au lieu de minimiser la taille totale de la formule, minimiser le nombre de variables partagées entre les sous-problèmes.

Pour les formules extrêmements larges, il peut-être impossible de pré-traiter la formule. Dans ce contexte là, nous considérons que le pré-traitement est aussi un défi pour les solveurs parallèles.

4.3.5 Échange de clauses apprises

Les solveurs SAT moderne génèrent des clauses apprises pour éviter de visiter le même espace de recherche. Les solveurs parallèles partagent/échangent ces clauses apprises entre les unités de calculs. Comme le nombre de clauses apprises peut être exponentiel dans le pire des cas et par conséquent ralentir la vitesse (la propagation unitaire) du solveur, des stratégies d'échanges ont été proposées pour communiquer efficacement entre les processeurs.

Parmi les stratégies les plus répandues, on peut citer par exemple celles basées sur l'échange de clauses ayant une taille inférieure à une limite fixée. Par exemple dans MANYSAT, seules les clauses de tailles inférieures à 8 sont échangées. Dans PLINGELING, le solveur ne partage que les clauses unitaires. Afin de permettre un échange tout le long de la recherche, dans Hamadi *et al.* (2009a), les auteurs ont proposé une stratégie dynamique pour incrémenter ou diminuer automatiquement la quantité de clauses partagées entre deux processeurs. En plus l'efficacité des clauses de petites taille fait de cette politique un critère simple mais efficace.

Dans Audemard *et al.* (2012), les auteurs ont proposé de renforcer l'efficacité d'échange, en combinant les deux critères LBD et PSM et la taille pour identifier les clauses à partager.

Á priori, la stratégies d'échanges entre les processeurs doivent être compatible avec la stratégie de réduction de la base. Dès lors adapter la stratégie de nettoyage aux différentes stratégies d'échange semble une voie intéressante à explorer.

4.4 Conclusion

Dans ce chapitre, nous avons présenté deux types d'algorithmes pour résoudre le problème SAT en parallèle sur des machines ayant une architecture multi-cœurs : l'approche diviser pour régner (en utilisant le chemin de guidage pour l'instant) et l'approche portfolio. D'une part, les solveurs de type portfolio semblent plus performants que les solveurs de type diviser pour régner selon les résultats des dernières compétitions. D'autre part, sur les instances très larges ou très difficiles, les approches diviser pour régner présentants de bonnes propriétés de scalabilité, sont plus intéressantes (une nouvelle approche de type diviser pour régner sera présenté dans le chapitre 8). Quelques pistes à explorer sont aussi discutées.

Conclusion

Le problème SAT est un problème central en intelligence artificielle, que ce soit d'un point de vue pratique (planification, vérification formelle, bio-informatique, etc) que d'un point de vue théorique (premier problème à avoir été montré NP-complet). D'un point de vue pratique les performances des solveurs n'ont cessé d'évoluer Le Berre (2010), permettant de résoudre des problèmes de plus en plus conséquents (des millions de variables et de contraintes). Cette évolution est due, conjointement, à l'avènement des solveurs SAT modernes Moskewicz *et al.* (2001) (*watched literal*, VSIDS, apprentissage) et à l'augmentation de la puissance de calcul (doublant tous les deux ans). Notre époque est témoin d'une « révolution » technologique, qui réside dans l'arrivée des architectures parallèles, permettant d'ouvrir de nouvelles perspectives.

Deux types d'approche pour résoudre SAT en parallèle sont principalement utilisés à l'heure actuelle. D'une part, les approches de type « diviser pour régner » consistent à partager l'arbre de recherche (« *chemin de guidage* ») Lewis *et al.* (2007). D'autre part, les approches de type portfolio Hamadi *et al.* (2009d) mettent les solveurs en concurrences, permettant ainsi de résoudre une formule à l'aide de différentes stratégies. Parmi elles, les approches de type portfolio sont démontrées empiriquement plus performantes à l'heure actuelle.

Dans la partie suivante, nous présentons nos contributions qui consiste à améliorer les performance des solveurs SAT parallèles.

Deuxième partie
Améliorations des prouveurs

Chapitre 5

Stratégies de la diversification et de l'intensification

Sommaire

DANS CE CHAPITRE, nous explorons les principes de diversification et d'intensification dans le cadre de la résolution parallèle du problème SAT de type portfolio. Les deux concepts qui jouent des rôles importants dans certains algorithmes de recherche y compris la recherche local, semblent être le point clé des solveurs SAT parallèles. Pour étudier ce compromis, nous définissons deux rôles pour les unités de calcul. Certaines d'entre elles, appelées maître, ont leur propres stratégies de recherche, et ont pour but d'assurer la diversification de la recherche. D'autres désignées comme « Esclaves » ont pour objectifs d'intensifier les stratégies de leur maîtres en explorant un sous-espace adjacent. Plusieurs questions importantes sont soulevées dans ce cadre. Quelles informations les maîtres devraient-ils envoyer aux esclaves pour intensifier la stratégie de recherche ? À quelle fréquence une unité subordonnée devrait recevoir ces informations ? Et finalement, quelle architecture maître-esclave est la meilleure ?

5.1 Introduction

Plusieurs solveurs SAT parallèles ont été proposés ces dernières années. Les premiers solveurs à voir le jour sont basés sur le principe « diviser pour régner ». Ces solveurs divisent soit l'espace de recherche en utilisant, par exemple, les chemins de guidages soit la formule elle même en utilisant des techniques de décomposition. Le problème majeur derrière ces approches à considérer, c'est le problème d'échange des clauses apprises entre les processeurs. Les autres solveurs SAT parallèles de type portfolio ont été étudiés récemment. Ils évitent le problème de distribution de la charge entre les processeurs en laissant plusieurs solveurs DPLL travailler ensemble en mode compétition/coopération pour être le premier à résoudre le problème. Chaque moteur (solveur séquentiel) dispose de la formule originale, l'espace de recherche n'est plus divisé ni décomposé. Afin d'être efficace, le portfolio doit utiliser des moteurs diversifiés (possédant des stratégies différentes). Cela permet d'éviter la redondance des sous-espaces visités. Cependant, afin de rendre l'échange de clauses plus efficace, la diversification doit être limité afin de maximiser l'impact de l'échange de clauses.

Une question donc importante est comment maintenir une bonne distance entre les espaces de recherche explorés par les différentes unités de calcul. Cela est équivalent à trouver un bon compromis entre la diversification et l'intensification. La réponse à cette question dépend fortement des problèmes. Pour les problèmes difficiles, il est préférable d'intensifier la recherche, alors que sur les instances plus faciles, la diversification permettrait de trouver une solution plus facilement.

Ce chapitre est organisé de la manière suivante. La première section détaille les différentes techniques pour intensifier une stratégie de recherche. La deuxième section explore les différents compromis de diversification/intensification dans un solveur de type portfolio. La troisième section présente les résultats expérimentaux. Nous finirons par une conclusion et certaines perspectives dans la quatrième section.

Les résultats reportés dans ce chapitre ont fait l'objet de la publication (Guo *et al.* (2010)).

5.2 Vers une bonne stratégie d'intensification

L'intensification de la recherche par une architecture maître/esclave consiste à permettre au maître de manipuler la recherche de l'esclave. Dans notre cas, cela consiste à imposer à l'esclave de se diriger vers un sous-espace de recherche en lui envoyant une interprétation partielle. Dès lors la question posée est quelle est l'interprétation partielle à envoyer à l'esclave et à quelle fréquence.

Tout d'abord, nous considérons un système composé de deux unités de calcul, respectivement un maître(M) et un esclave(S) (voir figure 5.1). Le rôle du maître est d'invoquer l'esclave pour l'intensification de sa recherche (le flèche bleu pointillé dans la figure 5.1). Par l'intensification, le but est de forcer l'esclave à explorer « différemment » l'espace de recherche autour de celui exploré par le maître. Les clauses apprises du maître et de l'esclave sont partagés dans les deux sens (la ligne noire dans la figure 5.1)

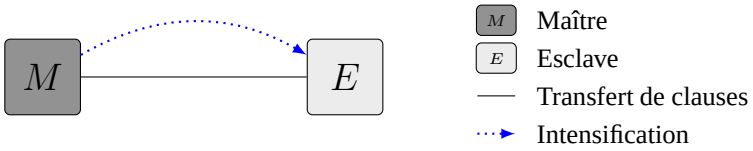


FIGURE 5.1 – Topologie d'intensification

Afin d'explorer différemment autour d'un espace de recherche donné, nous avons considéré différentes types d'informations. Supposons que le maître est actuellement dans l'état $S_M = (\mathcal{F}, \mathcal{D}_M, \Gamma_M)$, d'où $\mathcal{F}$ est l'instance originale, $\mathcal{D}_M$ est l'ensemble de décision, et Γ_M est la base des clauses apprises. Dans la suite, à partir d'un état donné, nous analysons les caractéristiques des trois informations que le maître peut transférer à l'esclave.

Nous utilisons la figure 8.1 pour illustrer ces caractéristiques. La figure représente un état courant S_M qui correspond aux branches vers le dernier conflit k. La liste des décisions dans la dernière branche est x_1, x_2,..., x_{n_k}. Les boites donnent une vue partielle du graphe d'implications obtenu sur les k dernier conflits générés après avoir affecté les dernières décisions x_{n_k}, $x_{n_{k-1}}$, ..., et x_{n_1}. Les clauses apprises sont respectivement $(\alpha_k \vee a_k)$, $(\alpha_{k-1} \vee a_{k-1})$, ..., et $(\alpha_1 \vee a_1)$ d'où a_k, a_{k-1}, ..., et a_1 sont les littéraux assertives correspond aux First-UIP $\neg a_k$, $\neg a_{k-1}$, ..., et $\neg a_1$.

5.2.1 La liste de décisions courante : decision list

La première information que nous avons testé est l'ensemble des décisions de l'interprétation courante $\mathcal{D}_M$(*decision list*). En utilisant cette liste de décisions, l'esclave peut construire l'ensemble ou un sous-ensemble de l'affectation partielle courante du maître sous la condition que toutes les clauses assertives générées par le maître soient envoyées à l'esclave. En choisissant de ne pas envoyer à l'esclave les activités des variables, on s'assure que ce dernier va explorer différemment l'espace de recherche.

5.2.2 L'ensemble de littéraux assertives : asserting set

La seconde information que nous avons utilisé est la séquence $A_M =< a_k, a_{k-1}, \ldots, a_1 >$(*asserting set*), l'ensemble de littéraux assertives des clauses apprises avant d'arriver à l'état courant S_M. Cette séquence est ordonné selon leur date de création (du dernier au premier conflit). En branchant sur la séquence A_M en utilisant la même polarité, l'esclave peut construire une affectation partielle contenant les littéraux assertives plus récents générés par le maître. Rappelons qu'un littéral a_i est dans la clause apprise $\alpha \vee a_i$ du maître. Comme l'esclave va brancher sur a_i, la future analyse du conflit sur a_i pourrait générer des clauses apprises contenant $\neg a_i$. Plus généralement, forcer l'esclave à brancher sur la séquence A_M va lui permettre de produire des clauses apprises plus pertinentes (des clauses apprises générées par l'esclave ont des littéraux complémentaires avec les clauses apprises générées par le maître et cela va renforcer la résolution entre eux). C'est évidemment un processus d'intensification, puisque les clauses apprises de l'esclave contiennent les littéraux les plus importants du maître, et grâce aux littéraux complémentaires entre les clauses apprises du maître et de l'esclave, une preuve de résolution plus intéressante pourrait être construite.

5.2.3 La séquence d'ensemble des littéraux conflits : conflict sets

La dernière information que nous avons utilisée est la séquence de l'ensemble des littéraux $C_M =< s_k, s_{k-1}, \ldots, s_1 >$ rencontrés pendant l'analyse de conflits du maître (*conflict sets*). L'ensemble s_k représente l'ensemble de littéraux rencontrés pendant l'analyse du dernier conflit. Plus précisément, les littéraux dans s_k correspondent aux nœuds situés entre le conflit et le First-UIP $\neg a_k$ dans le graphe d'implications (voir la figure 8.1). En plus, le littéral du conflit et le littéral $\neg a_k$ sont aussi dans s_k. On peut définir $s_k =< y_{k_1}, y_{k_2}, \ldots, y_{k_m} >$ où y_{k1} représente le littéral First-UIP $\neg a_k$ et y_{km} et correspondent au littéral du conflit comme il apparaît dans l'interprétation partielle. Le but d'envoyer cette séquence de l'ensemble de littéraux est d'intensifier la recherche en dirigeant l'esclave autour du même conflit.

Nous pouvons remarquer que, les séquences A_M et C_M pourraient contenir des littéraux redondants. Cela ne pose aucun problème, puisque l'esclave S affecte les littéraux dans l'ordre défini et il choisit toujours les littéraux non affectés.

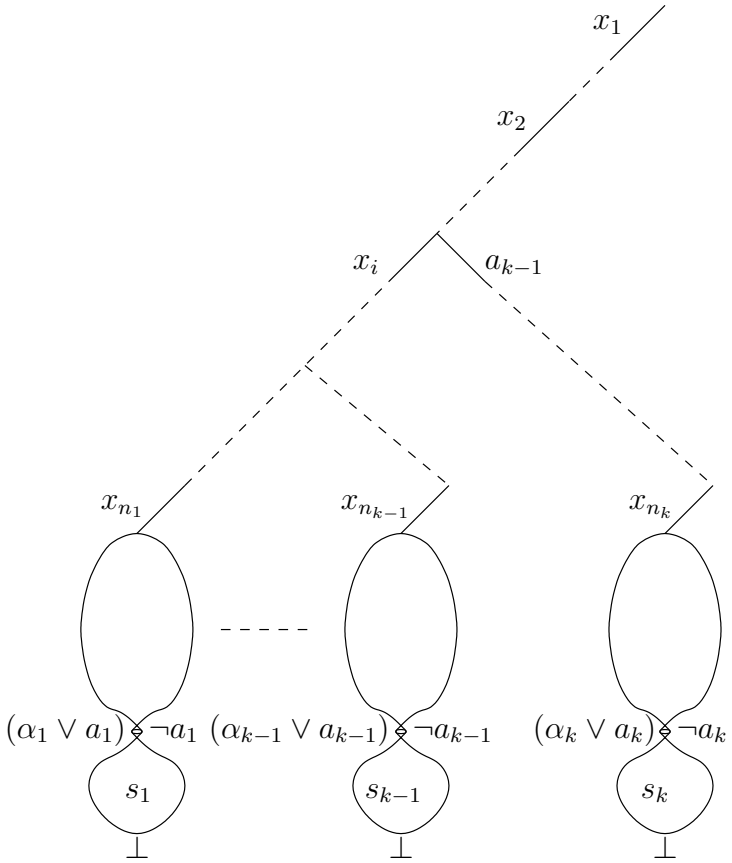


FIGURE 5.2 – Une vue partielle de l'arbre de recherche du maître

Afin de comparer la pertinences des stratégies d'intensifications définies précédemment, nous avons effectué les expérimentations suivantes sur les instances issues de la compétition SAT 2009, catégorie industrielle. Nous avons utilisé MANYSAT avec deux unités de calcul (voir figure 5.1), et les clauses apprises de taille inférieure ou égale à 8 sont partagées. Le maître M transfère à l'esclave E les informations lorsqu'il effectue un redémarrage. Pour le maître M. Afin de permettre à l'esclave d'être proche de son maître associée, la fréquence d'envoi de l'information de branchement est petite et constitue par conséquent la stratégie de redémarrage de l'esclave.

La figure 5.3 illustre la comparaison empirique en utilisant les trois stratégies d'intensification (*decision list*, *asserting set* et *conflict sets*). Nous pouvons observer que diriger la recherche en utilisant les *conflict sets* donne le meilleur résultat. Le nombre d'instances résolues en utilisant *decision list*, *asserting set* et *conflict sets* sont respectivement 201,207 et 212. Dans la suite de ce chapitre, nous allons utiliser *conflict sets* comme la stratégie d'intensification de référence.

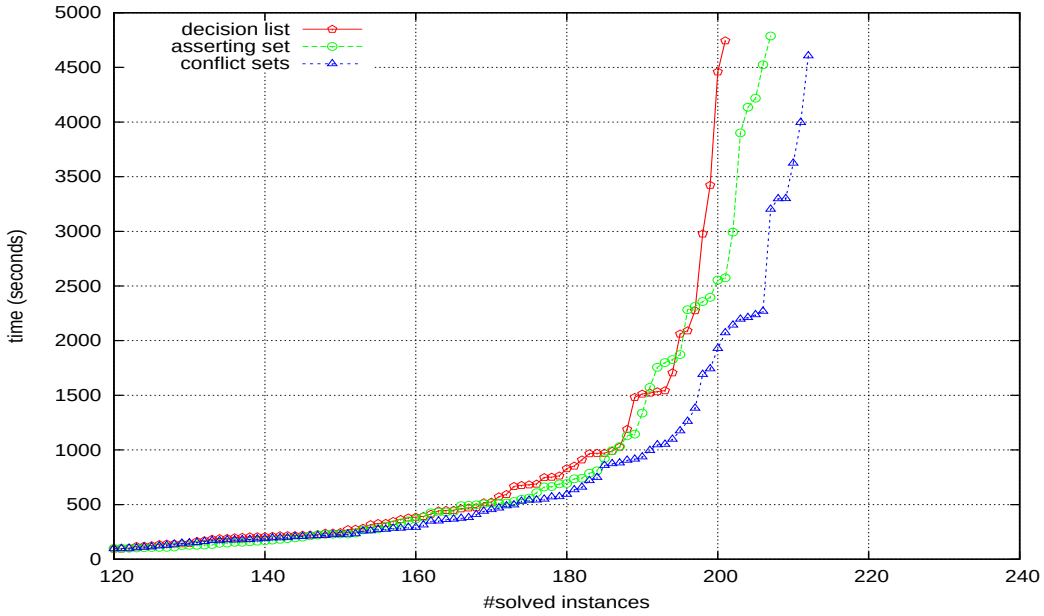


FIGURE 5.3 – La comparaison de trois stratégies d'intensification

5.3 Vers un compromis intensification/diversification

Dans cette section, nous allons explorer le compromis entre la diversification et l'intensification. Nous utilisons l'architecture de MANYSAT qui est représentée par une clique de quatre unités de calcul liés par l'échange des clauses apprises jusqu'à la taille 8. Ces unités représentent un ensemble de stratégies entièrement diversifiés. Afin de renforcer l'intensification, nous proposons d'étendre l'architecture et partitionner les unités de calcul entre les maîtres et les esclaves. Si nous permettons l'échanges des clauses apprises entre les esclaves, nous avons au total sept configurations possibles. Elles sont représentées dans la figure 5.4. Dans la figure, les lignes en pointillés représentent les relations entre les maîtres et esclaves. Remarquons que quand un maître dirige plusieurs esclaves, le maître alterne son guidage entre les esclaves *i.e.* à tour de rôle. En plus, quand une configuration contient une ou plusieurs chaîne(s) d'esclaves, (voir (d), (f) et (g) dans la figure), l'intensification d'un esclave de niveau i est déclenché par l'esclave de niveau $i - 1$.

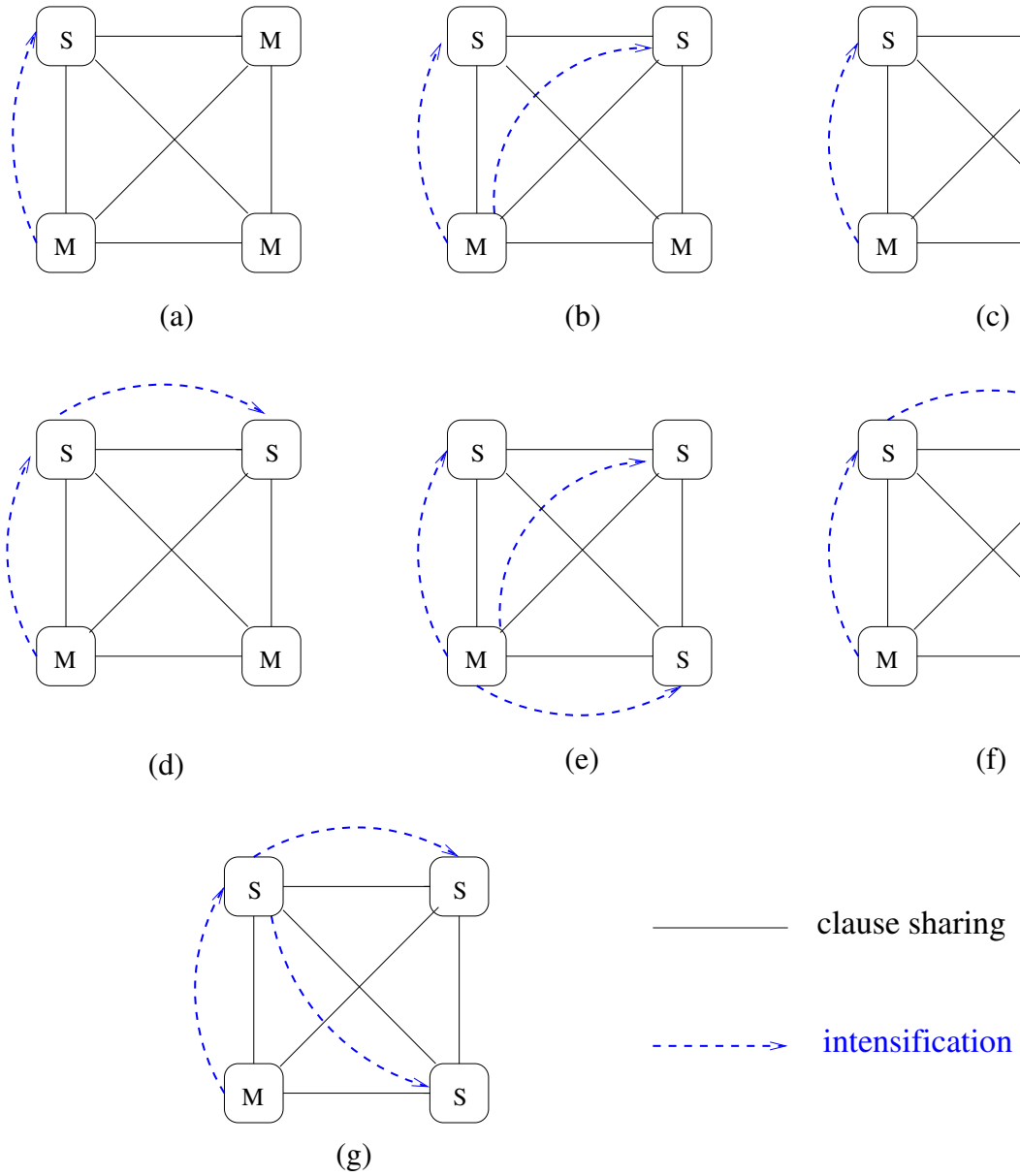


FIGURE 5.4 – Les topologies des diversification/intensification

Ces configurations représentent tous les compromis de diversification/intensification possible qu'on peut implémenter au dessus de l'architecture de MANYSAT. La section suivante présente leurs performances respectives y compris le solveur MANYSAT original.

5.4 Expérimentations

Afin de trouver la topologie la plus performante, nous testons dans cette section toutes les topologies présentées dans la section précédente (implémenté au dessus de l'architecture de MANYSAT) sur l'ensemble d'instances issues de la compétition SAT 2009, catégorie industrielle.

L'ensemble de nos tests sont effectués sur un cluster Quad-core Intel XEON X5550 avec 32Gb de

mémoire et 2.66 GHz de fréquence. Pour chaque instance, le temps CPU est limité à 4 heures qui correspond à 1 heure de temps limite pour chaque unité de calcul. Notre Maître/Esclave et leur configuration différentes sont implémentés sur MANYSAT classique. Ce solveur est aussi utilisé comme une base de comparaison. Nous avons utilisé *conflict sets* comme la stratégie d'intensification.

La table 5.1 résume nos résultats. La premier colonne présente les différentes méthodes, *i.e.*, le solveur MANYSAT classique (la première ligne) et MANYSAT étendu avec l'une de nos sept topologies de diversification/intensification (voir Figure 5.4). Dans la seconde colonne, le premier nombre représente le nombre total d'instances SAT résolues par les méthodes associées, le deuxième nombre (entre parenthèse) indique le nombre d'instances satisfiables résolues par l'esclave. La troisième colonne représente les informations similaires pour les problèmes insatisfiables. La colonne 4 indique le nombre total d'instances résolues, encore une fois, le nombre d'instances résolues par l'esclave est donné entre parenthèse. Finalement, les deux dernières colonnes présentent respectivement le temps total (cumulé) et le temps moyen en secondes. Le temps moyen est calculé sur l'ensemble des 292 instances. Une heure est utilisée pour les instances non résolues.

Cette table montre que la plupart des extensions basées sur nos topologies sont meilleures que MANYSAT classique. Ce solveur a résolu 212 instances tandis que la meilleure topologie résout 221. Remarquablement, toutes les topologies sont capables de résoudre plus de problèmes insatisfiables que MANYSAT. Ce résultat attendu montre que l'intensification est plus favorable pour les instances insatisfiables. En effet, notre stratégie d'intensification augmente la pertinence des clauses apprises échangées entre les maîtres et les esclaves. Comme les instances insatisfiables sont principalement résolues par résolution, l'amélioration de la qualité des clauses apprises améliore la performance sur les instances insatisfiables.

Méthode	**# SAT**	**# UNSAT**	**Total**	**Tot. temps (sc.)**	**Moy. temps**
ManySAT	87	125	212	329378	1128
Topo. (a)	86 (7)	133 (49)	219 (56)	311590	1067
Topo. (b)	84 (28)	130 (73)	214 (101)	324800	1112
Topo. (c)	**89** (23)	132 (74)	**221 (97)**	**307345**	**1052**
Topo. (d)	87 (25)	132 (67)	219 (92)	315537	1080
Topo. (e)	86 (45)	131 (109)	217 (154)	323208	1106
Topo. (f)	82 (44)	128 (102)	210 (146)	339677	1163
Topo. (g)	80 (45)	127 (107)	207 (152)	343800	1177

TABLE 5.1 – Compétition SAT 2009, Catégorie Industrielle : résultats globaux

D'après les résultats comparatifs obtenus par nos différentes topologies, il semble qu'équilibrer globalement le nombre de maîtres et d'esclaves est le choix le plus judicieux (topologie b, c et d). Parmi ces stratégies, associer à chaque mâitre un esclave se révèle la meilleure topologie (c).

La table 5.2 présente les résultats obtenus par notre meilleure topologie (c) comparativement à MANYSAT classique sur trois familles de problèmes. Nous pouvons constater que notre topologie (c) est meilleure que MANYSAT sur tous ces problèmes. Plus intéressant encore, notre algorithme permet de résoudre deux instances ouvertes (9dlx_vliw_at_b_iq8, and 9dlx_vliw_at_b_iq9), qui étaient prouvé comme insatisfiables pour la première fois en 2009.

La figure 5.5, présente la comparaison du temps cumulé entre MANYSAT classique et la topologie (c) sur l'ensemble des instances. Sur les instances faciles et moyennes (résolues en moins de 10 minutes), les deux algorithmes ont un comportement similaire. Cependant, quand les problèmes deviennent plus

Instance	**Statut**	**ManySAT**	**Topologie (c)**
9dlx_vliw_at_b_iq1	UNSAT	87.3	**10.6**
9dlx_vliw_at_b_iq2	UNSAT	226.3	**27.1**
9dlx_vliw_at_b_iq3	UNSAT	602.8	**103.2**
9dlx_vliw_at_b_iq4	UNSAT	1132	**163.5**
9dlx_vliw_at_b_iq5	UNSAT	2428	**313.1**
9dlx_vliw_at_b_iq6	UNSAT	–	**735.6**
9dlx_vliw_at_b_iq7	UNSAT	–	**991**
9dlx_vliw_at_b_iq8	**UNSAT**	–	**1822.7**
9dlx_vliw_at_b_iq9	**UNSAT**	–	**2670.1**
velev-pipe-sat-1.0-b10	SAT	4.4	**3.6**
velev-engi-uns-1.0-4nd	UNSAT	5	**4.9**
velev-live-uns-2.0-ebuf	UNSAT	6.7	**6.8**
velev-pipe-sat-1.0-b7	SAT	48.3	**6.2**
velev-pipe-o-uns-1.1-6	UNSAT	65.2	**30.8**
velev-pipe-o-uns-1.0-7	UNSAT	149.9	**118.2**
velev-pipe-uns-1.0-8	UNSAT	274.5	**82.7**
velev-vliw-uns-4.0-9C1	UNSAT	297.2	**235.4**
velev-vliw-uns-4.0-9-i1	UNSAT	–	**1311.6**
goldb-heqc-term1mul	UNSAT	23.8	**4.3**
goldb-heqc-i10mul	UNSAT	36.3	**23.5**
goldb-heqc-alu4mul	UNSAT	49.9	**40.9**
goldb-heqc-dalumul	UNSAT	384.1	**33.6**
goldb-heqc-frg1mul	UNSAT	2606	**83.1**
goldb-heqc-x1mul	UNSAT	–	**246.9**

TABLE 5.2 – Compétition SAT 2009, Catégorie Industrielle : résultats en temps (en secondes) sur trois familles de problèmes

difficiles, notre nouvelle architecture améliore sensiblement les performances, en résolvant 9 instances en plus.

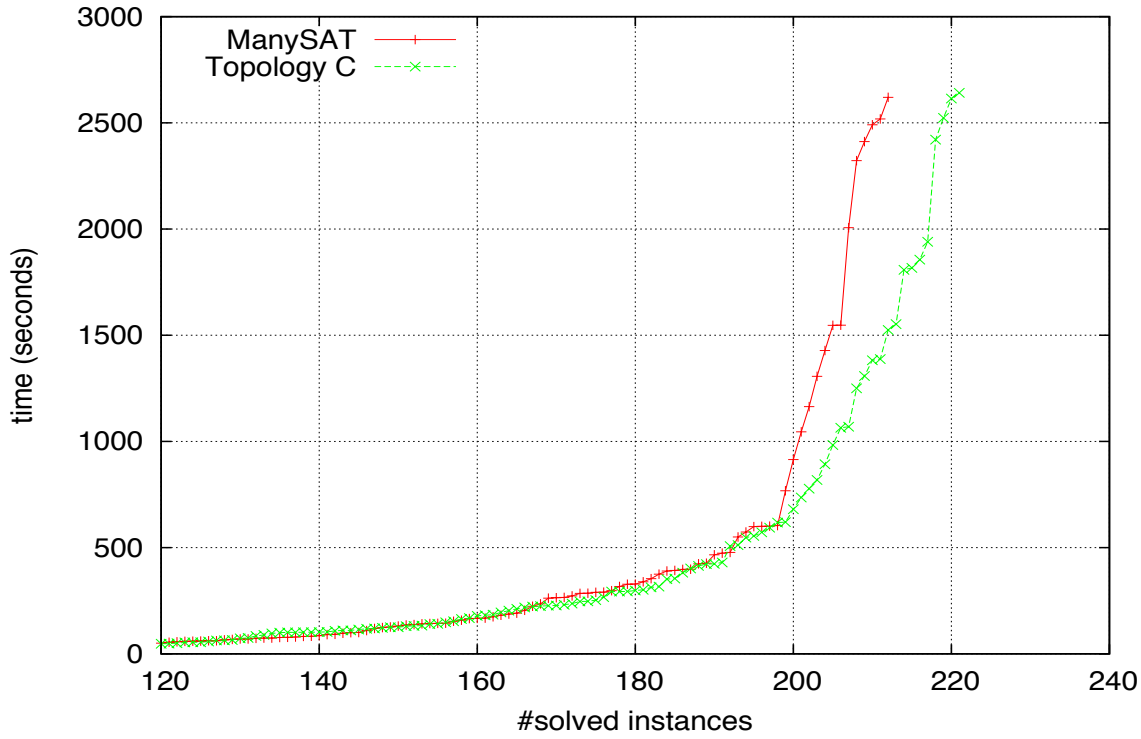


FIGURE 5.5 – SAT Compétition 2009, Catégorie Industriels : temps cumulé

5.5 Conclusion

Nous avons exploré les principes de diversification et d'intensification dans le cadre de la résolution parallèle du problème SAT de type portfolio. Les deux concepts jouent un rôle important dans certain algorithmes de recherche y compris la recherche local, et semblent être le point clé des solveurs SAT parallèles. Pour étudier le compromis entre la diversification et l'intensification, nous avons défini deux rôles pour les unités de calcul. Certaines d'entre elles avec leur propres stratégies de recherche, ont pour but d'assurer la diversification et sont classifiés comme « Maîtres ». Le restes, classées comme « Esclaves » ont pour but d'intensifier les stratégies de leurs maîtres.

Plusieurs questions importantes ont été soulevées et des réponses ont été apportées. La première, du coté maître, quelles informations devraient être envoyées aux esclaves pour intensifier la stratégie de recherche du maître ? Il semble que passer l'ensemble des littéraux collectés pendant la dernière analyse de conflit donne le meilleur résultat. Cette stratégie vise à diriger l'esclave vers les variables fortement liés aux conflits du maître, celui-ci permet aux maîtres et aux esclaves de partager les clauses les plus pertinentes mais dans des ordres différents.

La seconde question est à quelle fréquence une unité subordonnée devrait recevoir ces informations ? Nous avons décidé d'explorer la politique de redémarrage du maître et à mettre à jour les informations transférés aux esclaves. Nous avons montré que transférer à périodes courtes les informations aux esclaves, permet à l'esclave de rester connecté au sous-espace exploré par le maître.

Finalement, nous avons répondu à la question de trouver la meilleure structure Maître/Esclave. Nos expérimentations ont montré que équilibrer le nombre de maîtres et d'esclaves ainsi que le nombre d'esclaves du maître donne les meilleurs résultats. En particulier, notre meilleure topologie résout 9

instances en plus dans la catégorie industrielle par rapport au meilleur solveur MANYSAT. Les résultats ont montré aussi l'intérêt de l'intensification sur les problèmes insatisfiables. Remarquablement, notre nouvelle stratégie permet de résoudre deux instances ouvertes en 2009.

En perspective, nous souhaitons adapter dynamiquement le nombre de maître/esclave suivant l'instance à résoudre. Nous aurons ainsi une architecture dont la topologie est évolutive au cours du temps. Nous pensons que l'aspect adaptatif sera plus efficace qu'une topologie statique.

Chapitre 6

Ajustement dynamique de l'heuristique de polarité

Sommaire

DANS CE CHAPITRE, nous proposons une nouvelle heuristique pour la polarité d'affectation d'une variable, dans le cadre d'un solveur SAT parallèle (voir chapitre 4). La polarité selon laquelle le prochain point de choix sera affecté est un processus important des solveurs SAT modernes, en particulier pour les solveurs de type portfolio. En effet, ces solveurs sont souvent basés sur une approche de type coopération/compétition. Dans ce cas, la polarité peut être utilisée pour diriger le solveur dans l'espace de recherche. Nous proposons un critère basé sur le *progress saving* afin d'évaluer si deux solveurs explorent le même espace de recherche. Une fois ce critère établi, nous l'utilisons pour attribuer une heuristique de polarité aux différents solveurs de manière dynamique. Des résultats prometteurs ont été obtenus et montrent l'intérêt de notre approche sur l'affectation de polarité dynamique dans le cadre de solveurs SAT parallèles de type portfolio.

6.1 Introduction

Les solveurs SAT modernes sont très sensibles aux réglages de leurs paramètres initiaux Biere *et al.* (2009). Par exemple, sur certaines instances, changer les paramètres relatifs à la fréquence des redémarrages ou à l'heuristique de choix de polarité peut fortement affecter les performances d'un solveur. Dans ce contexte, une approche de type portfolio permet d'exécuter différentes versions d'un même solveur sur une même instance, profitant ainsi au maximum des capacités des solveurs SAT modernes.

Afin d'obtenir une approche portfolio efficace, il est nécessaire que les différents paramètres utilisés pour configurer les solveurs tendent à rendre les solveurs complémentaires entre eux. Une des difficultés de ce type d'approche est de paramétrer les différents solveurs de telle manière que deux solveurs n'effectuent pas la même tâche. En effet, dans ce cas, l'une des deux unités de calcul effectue un travail inutile et redondant. Le problème est qu'il est impossible de prévoir, *a priori*, le comportement d'un

solveur par rapport à ses paramètres initiaux. Pour pallier à ce problème, nous proposons dans cet article une mesure permettant d'estimer le comportement d'un solveur vis-à-vis d'un autre. Cette mesure est basée sur la notion de *progress saving* Pipatsrisawat et Darwiche (2007). L'intuition est que l'interprétation représentant le *progress saving* peut être vue comme une image de la recherche en cours et une intention d'affectation pour la suite de recherche. De cette manière, deux solveurs peuvent être considérés comme proches s'ils explorent de la même manière l'espace de recherche. Une fois cette mesure définie, celle-ci est utilisée pendant la recherche pour ajuster dynamiquement l'heuristique de choix de polarité. Cet ajustement a pour but d'éloigner deux solveurs considérés comme trop proches.

Le chapitre est organisé de la façon suivante. Nous présentons d'abord notre mesure permettant d'estimer si deux solveurs effectuent un travail redondant. Une fois notre mesure définie, nous étudions de manière expérimentale la distance entre les différents solveurs présents dans MANYSAT 1.1. Après analyse des résultats nous proposons un schéma d'ajustement de l'heuristique de polarité. Finalement, nous étudions expérimentalement l'impact de l'application d'un tel schéma dans le cas de trois approches *portfolio* (MANYSAT 1.1, le solveur *portfolio* appelé MANYIDEM qui utilise des solveurs séquentiels possédant une architecture identique et MANYSAT 1.5).

Cette contribution a donné lieu à deux publications Guo et Lagniez (2011a;b). Une partie de résultats est aussi présenté dans Lagniez (2011).

6.2 Estimer la distance entre deux solveurs

Comme précisé précédemment, les solveurs parallèles de type *portfolio* sont souvent basés sur le principe de compétition/coopération. Ainsi dans le solveur MANYSAT, la phase de coopération peut être assimilée au transfert de clauses apprises. Tandis que la phase de compétition peut être associée aux différentes stratégies choisies (redémarrage, heuristique de polarité, *etc*.) sur chacun des solveurs. Contrairement à la partie coopérative, la manière dont les solveurs entrent en compétition est choisie de manière statique au début de la recherche. Ce type d'approche ne permet ni d'identifier ni de traiter le cas où deux solveurs effectuent le même travail. En effet, même si deux solveurs ont des stratégies différentes, il n'est pas sûr qu'ils entrent en compétition. Il peut même arriver des cas où l'opposé se produit, c'est-à-dire que les deux solveurs vont entrer en phase de coopération. Cette coopération se traduit par un déséquilibre entre les deux phases et donc le plus souvent par un travail redondant de la part d'un des deux solveurs.

Pour éviter ce genre de situation, et ainsi préserver un schéma de type compétition/coopération, une méthode permettant d'ajuster dynamiquement l'heuristique de choix de la polarité est proposée. Cette méthode consiste à d'abord estimer si deux unités de calcul sont ou vont entrer dans la même espace de recherche, si c'est le cas, notre méthode va essayer les éloigner en ajustant dynamiquement leur heuristiques de choix de la polarité. Pour faire cela, une mesure permettant d'estimer la distance entre deux solveurs est d'abord introduite. Puis, afin d'étudier le comportement des solveurs vis-à-vis de notre mesure, nous avons effectué un ensemble d'expérimentations.

6.2.1 Distance entre deux solveurs

Afin de savoir si deux solveurs sont en train d'effectuer ou vont effectuer le même travail, nous définissons la notion d'intention comme étant l'interprétation complète obtenue à partir de l'heuristique de choix de polarité associée à un solveur.

Définition 6.1. *Soient $\mathcal{F}$ une formule* CNF *et S un solveur utilisant une heuristique de choix de polarité définie par la fonction f. L'intention est définie comme :*

$$\mathcal{P} = \{y \in \mathcal{L}_\mathcal{F} \text{ tel que } f(x) = y \text{ et } x \in \mathcal{V}_\mathcal{F}\}.$$

L'interprétation complète $\mathcal{P}$ peut d'une certaine manière être vue comme l'espace de recherche que le solveur souhaite atteindre. De cette manière, nous pouvons supposer que deux solveurs sont proches dans l'espace de recherche s'ils ont l'intention d'explorer le même espace. À partir de la notion d'intention, il est possible de définir une mesure permettant d'estimer la distance entre deux solveurs. Cette mesure est noté κ.

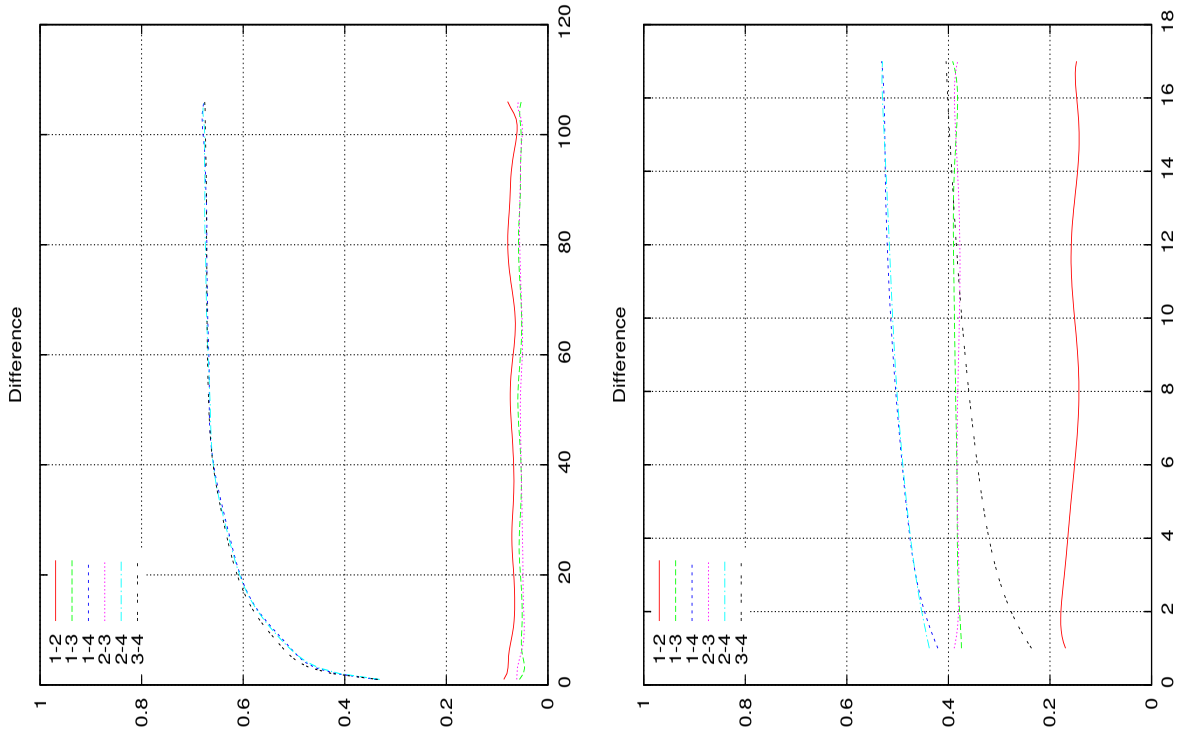
Définition 6.2. *Soient Σ une formule* CNF*, ($\mathcal{C}_i$, $\mathcal{P}_i$) et ($\mathcal{C}_j$, $\mathcal{P}_j$) deux solveurs $\mathcal{C}_i$ et $\mathcal{C}_j$ avec leurs intentions respectives $\mathcal{P}_i$ et $\mathcal{P}_j$. La distance entre deux solveurs, notée κ, est alors définie comme :*

$$\kappa(\mathcal{C}_i, \mathcal{C}_j) = 1 - \frac{|\mathcal{P}_i \cap \mathcal{P}_j|}{|\mathcal{V}_\mathcal{F}|}.$$

Remarquons que deux solveurs sont proches, d'après notre mesure, si leurs interprétations représentant leurs intentions respectives ont une faible distance de Hamming.

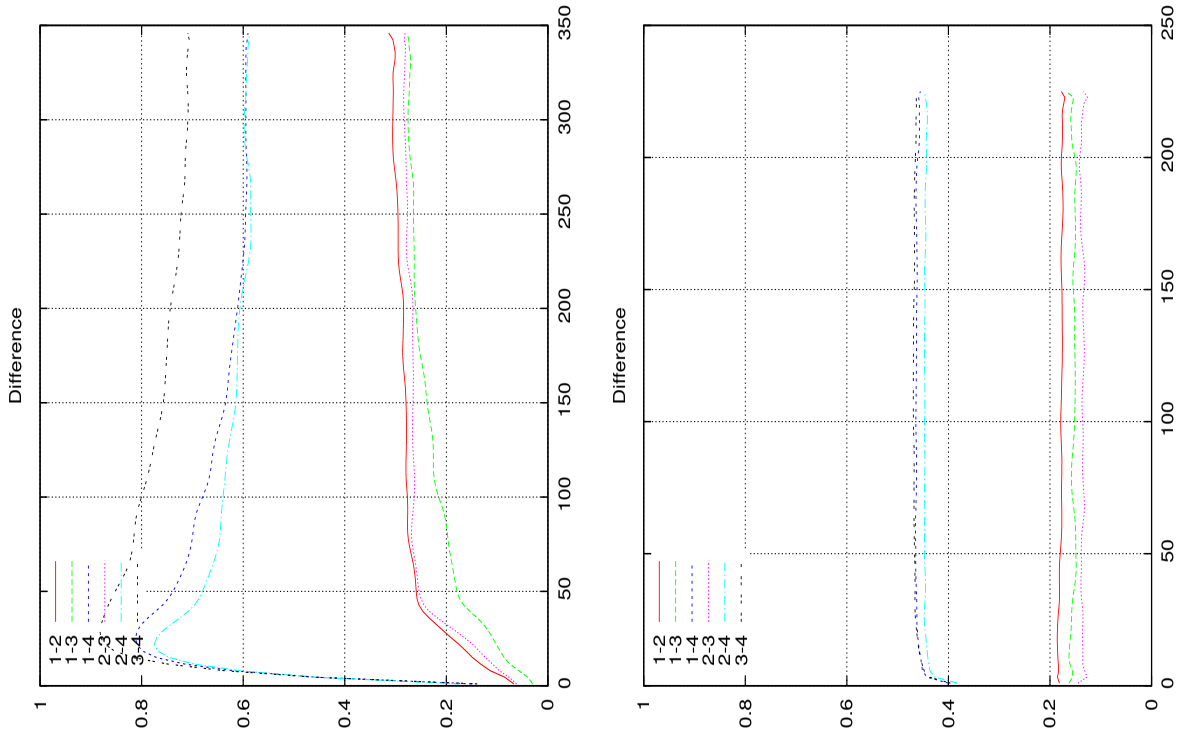
6.2.2 Évolution de la distance entre différents solveurs

Afin d'étudier l'évolution de la distance entre différents solveurs deux à deux, nous avons exécuté le solveur MANYSAT 1.1 (voir 4.2.2 pour la description) en récupérant la distance entre les différents solveurs tous les 5000 conflits. Les courbes de la figure 6.1 retranscrivent, de manière représentative, le comportement des différents solveurs les uns envers les autres.



(a) 9dlx_vliw_at_b_iq2

(b) schup-l2s-guid-1-k56



(c) vange-col-abb313GPIA-9-c

(d) countbitsarray08_32

FIGURE 6.1 – Comparaison deux à deux de la distance entre différents solveurs exécutés en parallèle. L'axe des abscisses représente le nombre de conflits atteint (divisé par 5000) et l'axe des ordonnées représente le résultat de notre mesure. Pour une meilleure lisibilité toutes les courbes ont été lissées.

Nous pouvons constater que les solveurs se divisent en deux catégories :

- la première catégorie regroupe les courbes où le cœur 4 n'apparaît pas, c'est-à-dire les courbes entre les cœurs 1-2, 1-3 et 2-3. Nous pouvons voir que les cœurs 1, 2 et 3 sont très proches les uns des autres. Ceci est facilement explicable pour les cœurs 1 et 2, étant donné qu'ils utilisent tous deux la même heuristique de choix de polarité (*progress saving*). En ce qui concerne le cœur 3, nous pensons que cela est dû au choix de la polarité initiale pour le *progress saving* (la phase étant affectée à « faux » au départ) ;
- la seconde catégorie regroupe les courbes où le cœur 4 apparaît. Nous observons que la distance entre le cœur 4 et n'importe quel autre cœur (courbes 1-4, 2-4, 3-4) est toujours supérieure au cas où le cœur 4 n'est pas présent dans le calcul de la distance (courbes 1-2, 1-3, 2-3). Ceci peut en partie être expliqué par le choix de la stratégie d'affectation de polarité du cœur 4. En effet, si nous considérons les unités de calcul 1 et 2, nous pouvons noter que lorsqu'ils apprennent une nouvelle clause α par analyse de conflit, elle est falsifiée par l'interprétation courante $\mathcal{I}$ (correspondant au *progress saving* du solveur, c'est-à-dire à son intention). Sans perte de généralité, supposons que le cœur 1 apprend la clause α. Cette clause est ensuite transférée au cœur 4 et le nombre d'occurrences des littéraux appartenant à α est incrémenté. Puisque l'heuristique de choix de polarité du cœur 4 est basée sur le nombre d'occurrences des littéraux, l'interprétation représentant son intention va avoir tendance à satisfaire les clauses proposées par les cœurs 1 et 2. Par conséquent, le solveur 4 s'éloigne du cœur 1 à chaque fois qu'il récupère une de ses clauses. En ce qui concerne le cœur 3, étant donné que son intention est proche de celle des cœurs 1 et 2, il est facile de comprendre pourquoi la distance qui le sépare du cœur 4 est grande.

6.2.3 Ajustement dynamique de la polarité

Après l'analyse du comportement des différents solveurs entre eux, nous avons choisi de définir une nouvelle heuristique de polarité. Puisque les solveurs peuvent être très proches dans l'espace de recherche nous avons décidé d'ajouter du « bruit » pour les éloigner les uns des autres. Pour cela, lorsque deux cœurs sont détectés comme « trop proches », nous choisissons d'inverser la polarité d'un des deux cœurs. Pour éviter que deux solveurs proches n'inversent en même temps leur polarité, l'ajustement se fait de manière unidirectionnelle, c'est-à-dire qu'un seul des deux solveurs ajuste sa polarité. De plus, afin de ne pas ralentir de manière excessive la vitesse du solveur, cet ajustement ne sera pas fait de manière systématique. La figure 6.2 décrit le transfert de messages et l'ajustement de la polarité entre deux solveurs. À chaque point de contrôle, représentés par des losanges, le cœur j demande au cœur i s'ils sont proches. Le cœur i lui répond par « oui » ou par « non » en fonction du résultat obtenu en mesurant sa distance avec le cœur j à l'aide de la mesure définie précédemment. Dans le cas où la réponse est négative, le cœur j utilise son heuristique de choix de polarité initiale. Dans le cas d'une réponse positive, le cœur j utilise son heuristique de polarité de manière inversée. Par exemple, supposons que le cœur b ait comme heuristique de polarité l'heuristique *false*. Dans ce cas, tant qu'un nouveau point de contrôle n'est pas atteint, le solveur affecte les prochains points de choix à vrai. Au prochain point de contrôle, l'heuristique de polarité peut de nouveau être réajustée. Ce processus est répété tant qu'une solution n'a pas été trouvée ou que le problème n'a pas été réfuté.

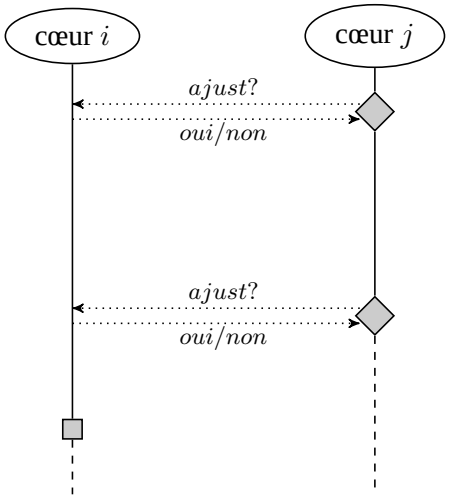


FIGURE 6.2 – Ajustement dynamique de la polarité d'un solveur vis-à-vis d'un autre.

6.3 Expérimentations

Afin d'étudier l'impact de cet ajustement, nous avons conduit les expérimentations suivantes. Dans un premier temps, nous avons choisi d'isoler l'heuristique de choix de polarité vis-à-vis des autres paramètres du solveur. Pour cela, hormis l'heuristique de choix de polarité, tous les cœurs utilisent des architectures identiques (politique de redémarrage, analyse de conflits et heuristique de choix de variables). Dans la seconde, nous avons intégré directement notre méthode au solveur MANYSAT 1.1. Enfin, nous étudions l'apport de notre schéma d'ajustement dans le solveur MANYSAT 1.5.

Avant d'entamer les expérimentations, il nous a fallu définir un schéma d'application pour notre méthode. En effet, dans la section précédente nous ne définissons pas formellement le cadre selon lequel notre ajustement doit être effectué.

Tout d'abord, à la vue des résultats obtenus concernant l'évaluation de la distance entre les solveurs pris deux à deux, notre méthode n'est pas appliquée sur le cœur 4 (cœur utilisant l'heuristique de polarité *occurence*). Ensuite, afin d'éviter des problèmes de cycle dans le protocole d'ajustement de polarité notre approche n'est pas non plus appliquée au cœur 1. La figure 6.3 schématise la manière selon laquelle l'ajustement des cœurs 2 et 3 est effectué. Nous avons choisi d'ajuster le cœur 2 et le cœur 3 dans le cas où ils sont proches du cœur 1, c'est-à-dire lorsque $\delta(\mathcal{C}_1, \mathcal{C}_2) \leq 0.1$ ou $\delta(\mathcal{C}_1, \mathcal{C}_3) \leq 0.1$ et que $\delta(\mathcal{C}_1, \mathcal{C}_2) > 0.1$. En ce qui concerne les points de contrôle, nous avons choisi de les effectuer tous les 5000 conflits. Cette valeur a été fixée expérimentalement.

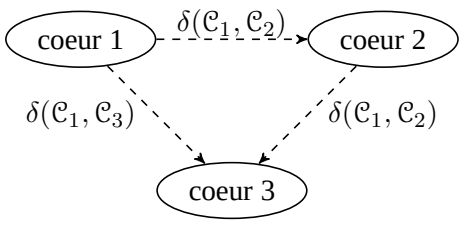


FIGURE 6.3 – Schéma d'ajustement MANYSAT 1.1.

L'ensemble des résultats expérimentaux reportés dans cette section ont été obtenus sur un Quad-core Intel XEON X5550 avec 32Gb de mémoire. Le temps CPU est limité à 900 secondes par unité de calcul. Pour ces expérimentations nous avons utilisé les 292 instances industrielles de la compétition SAT 2009 Le Berre et Roussel (2009). Toutes les instances sont pré-traitées à l'aide de SATELITE Eén et Biere (2005). Du fait du non déterminisme des solveurs parallèles, chaque solveur a été lancé trois fois sur toutes les instances. Les résultats reportés concerne l'exécution qui a résolu le plus d'instances (protocole utilisée lors de la compétition SAT de 2009).

6.3.1 Ajustement de l'heuristique de choix de polarité de MANYIDEM

Comme précisé précédemment, afin d'isoler l'heuristique de choix de polarité des autres composantes des solveurs SAT modernes, nous avons choisi d'utiliser un solveur, nommé MANYIDEM, avec une architecture identique pour l'ensemble des cœurs. Chaque unité de calculs utilise une politique de redémarrage définie par la suite de $luby(128)$, l'heuristique de choix de variable est VSIDS, le schéma d'apprentissage est le first-UIP et les clauses sont échangées si leur taille est inférieure à 8. En ce qui concerne l'heuristique de choix de polarité nous avons conservé celle implémentée dans la version initiale de MANYSAT 1.1 (voir tableau 4.2). MANYIDEM résout 72 instances satisfiables et 111 insatisfiables tandis que MANYIDEM avec notre technique d'ajustement résout 74 instances satisfiables et 120 insatisfiables. La figure 6.4 donne le nombre d'instances résolues (axe des abscisses) en fonction du temps (axe des ordonnées) par MANYIDEM avec et sans méthodes d'ajustement.

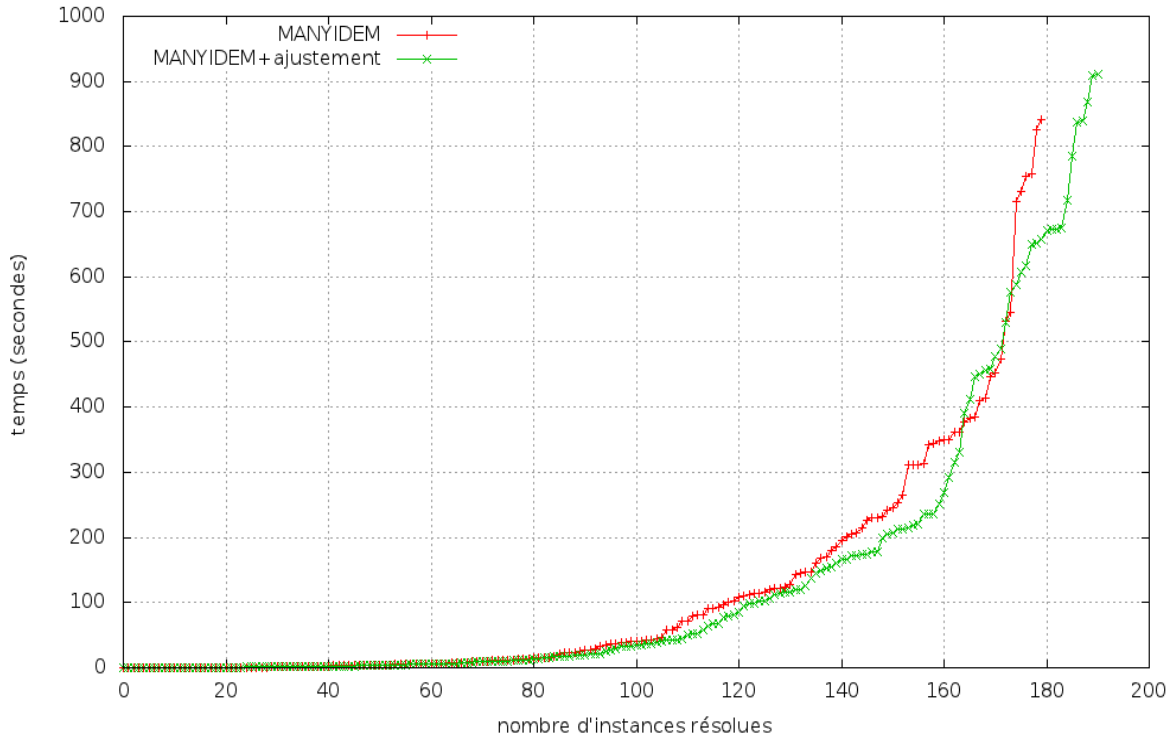


FIGURE 6.4 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle avec architecture identique intégrant ou pas notre technique d'ajustement d'heuristique de choix de polarité.

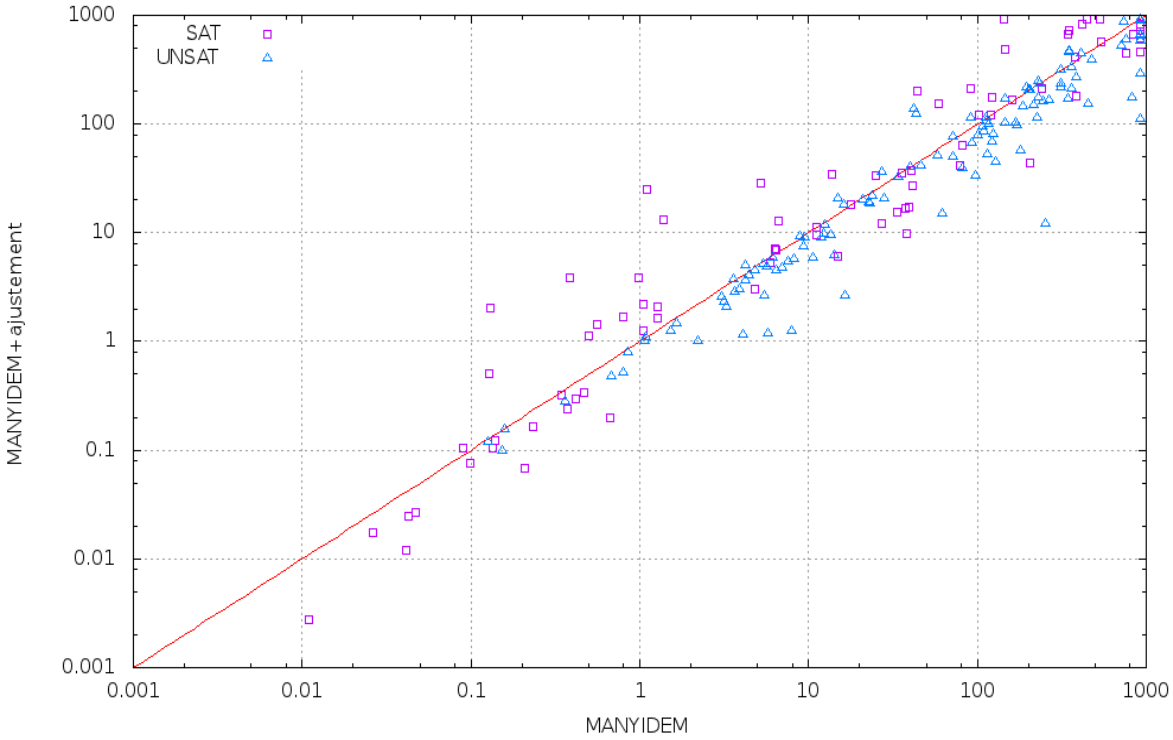


FIGURE 6.5 – Corrélation entre le temps mis par chacune des deux méthodes (MANYIDEM et MANYIDEM + ajustement) pour résoudre une instance donnée. Un point (temps MANYIDEM, temps MANYIDEM + ajustement) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

À la vue du nombre d'instances résolues (183 pour MANYIDEM et 194 pour MANYIDEM avec notre technique d'ajustement), nous pouvons conclure que notre technique permet d'améliorer les performances du solveur. De plus, comme le montre le nuage de points de la figure 6.5, l'ajout de notre méthode permet toujours de résoudre de manière plus efficace les instances insatisfiables. Concernant les instances satisfiables nous pouvons noter que, même si les points sont cette fois ci plus éparpillés, notre approche est plus compétitive.

6.3.2 Ajustement de l'heuristique de choix de polarité de MANYSAT 1.1

Dans cette partie, nous évaluons le gain apporté par notre technique d'ajustement de polarité dans le cadre du solveur MANYSAT 1.1 Hamadi *et al.* (2009d). Ce dernier résout 73 instances satisfiables et 120 instances insatisfiables. Les figures 6.6 et 6.7 reportent les résultats obtenus par MANYSAT 1.1 utilisant ou non notre technique d'ajustement de la polarité. Ces dernières montrent clairement que l'ajout de notre méthode au sein du solveur SAT parallèle MANYSAT 1.1 permet d'améliorer sa compétitivité (81 instances satisfiables et 123 instances insatisfiables résolues). Si nous étudions plus finement les résultats à l'aide du nuage de points, nous nous rendons compte que comme dans le cas où les architectures sont identiques, notre méthode permet au solveur de résoudre plus efficacement les instances insatisfiables (les points ont tendance à se trouver en dessous de la diagonale). En ce qui concerne les instances satisfiables, les résultats expérimentaux montrent que notre approche permet de résoudre sensiblement plus d'instances (plusieurs points agglutinés sur la droite $x = 900$ signifiant que le solveur MANYSAT 1.1 sans notre technique n'a pas été capable de résoudre).

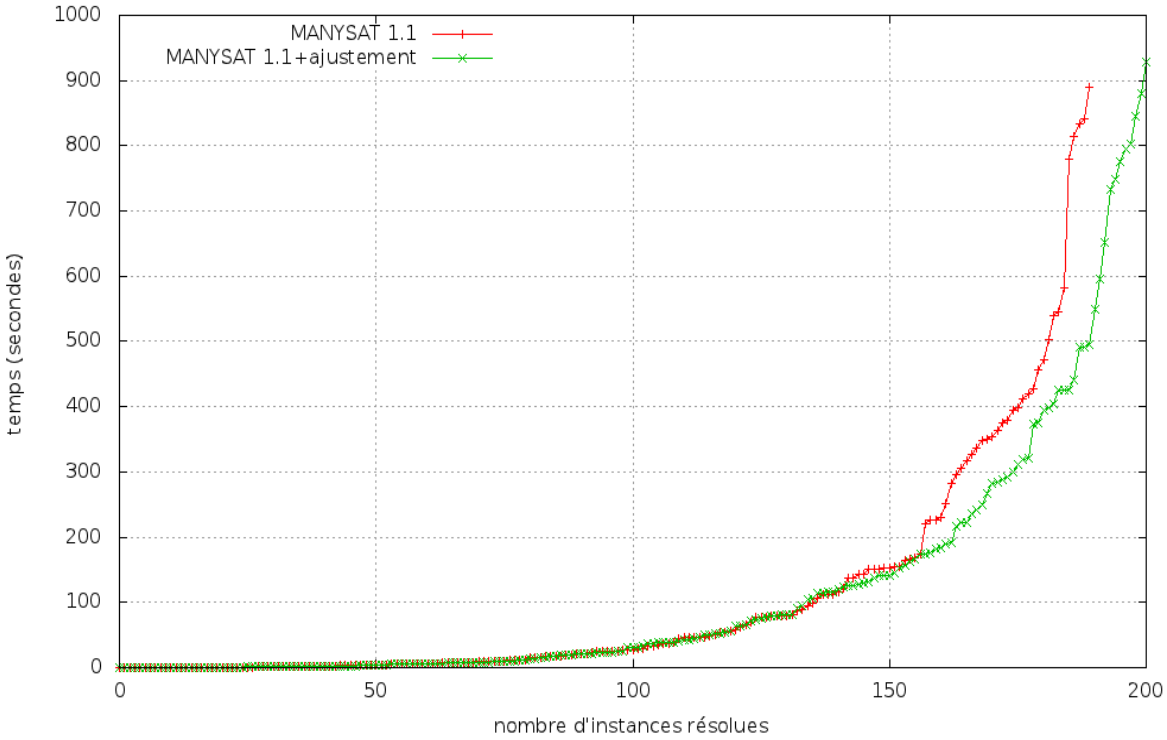


FIGURE 6.6 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle MANYSAT 1.1 intégrant ou pas notre technique d'ajustement de polarité.

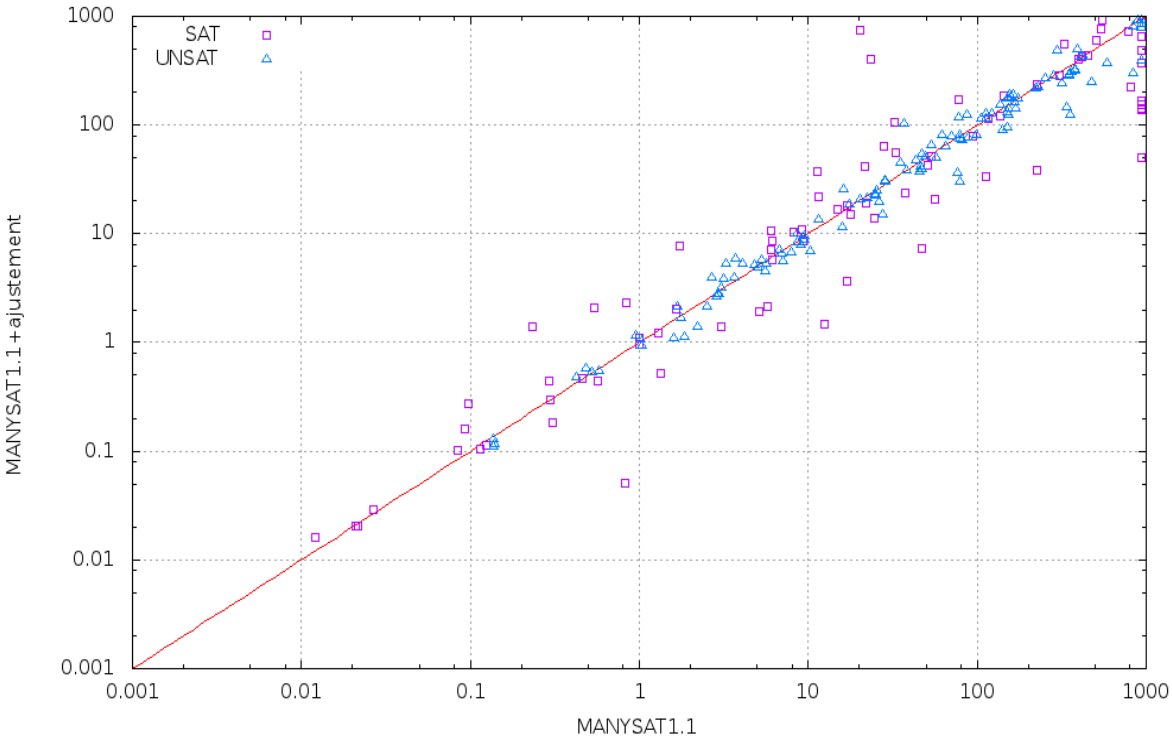


FIGURE 6.7 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT 1.1 et MANYSAT 1.1 + ajustement) pour résoudre une instance donnée. Un point reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

6.3.3 Ajustement de l'heuristique de choix de polarité de MANYSAT 1.5

MANYSAT 1.5 Guo *et al.* (2010) est une extension de MANYSAT 1.1 dans laquelle deux cœurs maîtres (cœur $\mathcal{M}1$ et cœur $\mathcal{M}2$) invoquent périodiquement deux esclaves (cœur $\mathcal{E}1$ et cœur $\mathcal{E}2$) dans le but d'intensifier une certaine partie de l'espace de recherche (voir le chapitre 5). Ce solveur est basé sur le principe d'intensification/diversification. Afin d'évaluer la robustesse de notre approche nous avons incorporé notre stratégie d'ajustement de polarité au sein de MANYSAT 1.5. Néanmoins, puisque la structure du solveur est différente de celle de MANYSAT 1.1, nous avons défini une nouvelle topologie d'ajustement. Cette dernière, reportée sur la figure 6.8, consiste à ne considérer que les unités de calcul maître (c'est-à-dire $\mathcal{M}1$ et $\mathcal{M}2$) dans le processus d'ajustement. Nous avons choisi d'ajuster le cœur $\mathcal{M}2$ dans le cas où il est proche du cœur $\mathcal{M}1$ ($\delta(\mathcal{M}1, \mathcal{M}2) < 0.1$).

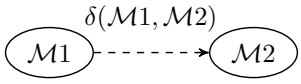


FIGURE 6.8 – Schéma d'ajustement de MANYSAT 1.5.

Les figures 6.9 et 6.10 illustrent les résultats obtenus par MANYSAT 1.5 (79 instances satisfiables et 124 instances insatisfiables résolues) et MANYSAT 1.5 avec notre technique d'ajustement (79 instances satisfiables et 127 instances insatisfiables résolues). Malgré le fait que l'ajout de notre méthode ne permet pas d'améliorer sensiblement les performances du solveur MANYSAT 1.5, nous pouvons observer qu'en ce qui concerne la résolution des instances insatisfiables la tendance précédemment observée se confirme. En effet, le nuage de point (figure 6.10) indique que la plupart des points représentant les instances insatisfiables se trouvent sous la diagonale.

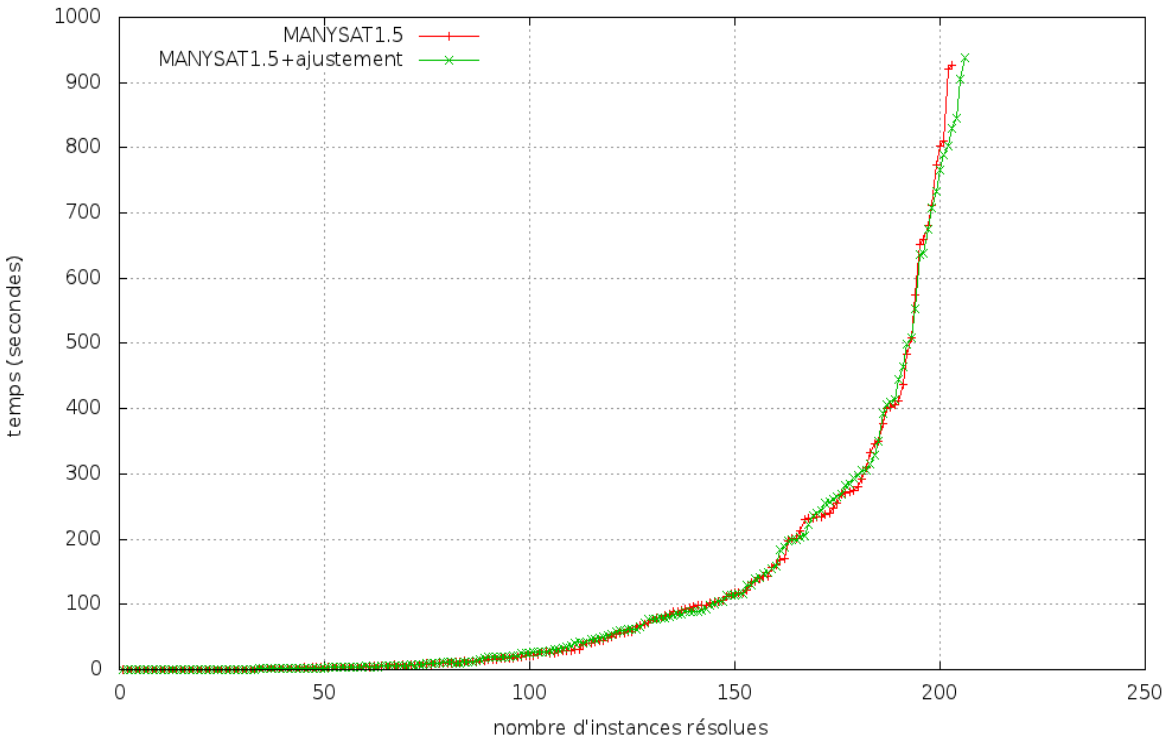


FIGURE 6.9 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle MANYSAT 1.5 intégrant ou pas notre technique d'ajustement de polarité.

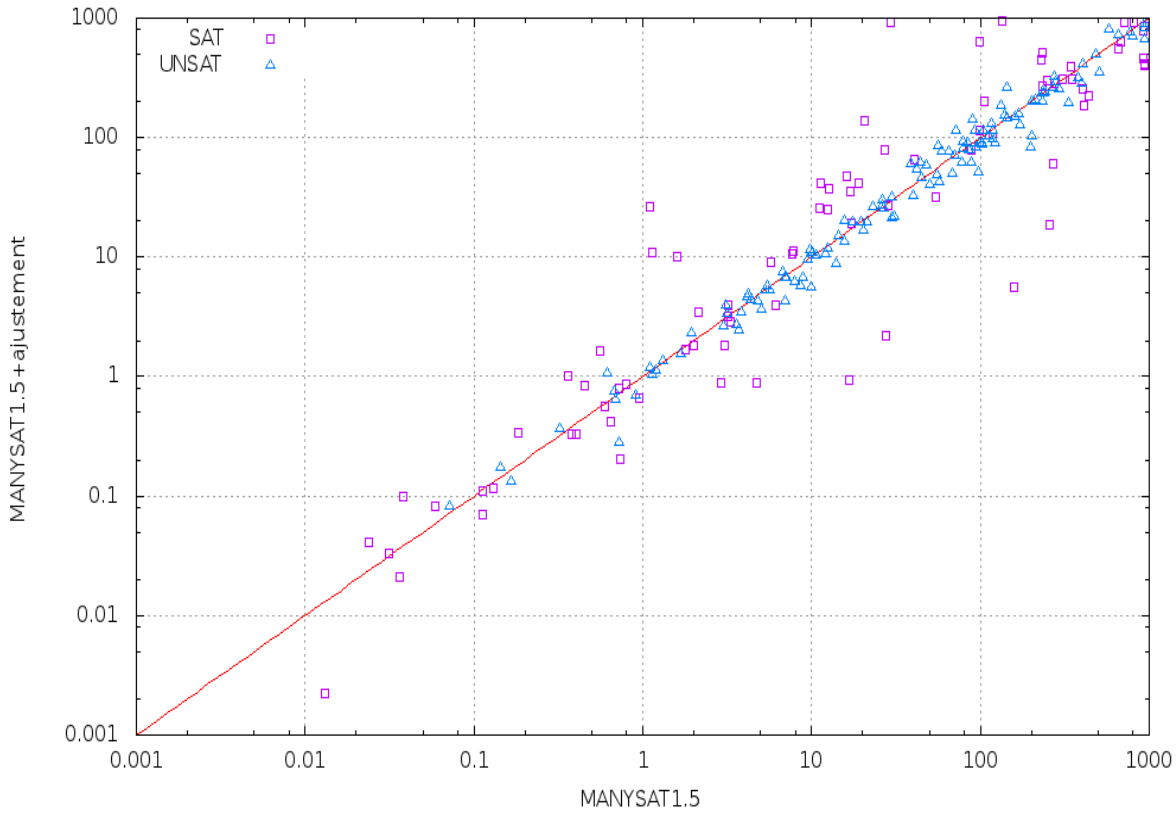


FIGURE 6.10 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT 1.5 et MANYSAT 1.5 + ajustement) pour résoudre une instance donnée. Un point reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

6.3.4 Résultats classés par famille d'instances

Les résultats expérimentaux reportés dans les sections précédentes ont permis de mettre en exergue le fait que l'ajout de notre stratégie d'ajustement dynamique de choix de polarité au sein de différentes versions du solveur MANYSAT permettait d'améliorer globalement leur efficacité. Dans cette partie, nous tentons d'étudier plus finement le comportement de notre approche en analysant les résultats obtenus sur certaines familles d'instances.

Les Tables 6.1 et 6.2 reportent les résultats obtenus sur certaines familles de problèmes sélectionnées parmi l'ensemble des instances industrielles de la compétition 2009. Les configurations de MANYSAT testées sont notées avec un « + » lorsque notre technique d'ajustement a été implémentée. Nous pouvons tout d'abord observer que notre technique d'ajustement permet d'améliorer les performances de MANYSAT sur plusieurs classes d'instances. De plus, il faut noter que, dans la plupart des cas, lorsque notre approche permet d'améliorer les performances d'une version de MANYSAT, elle améliore aussi les autres versions (vliw_unsat_2.0, q_query_3, AProVE, ...). Cette particularité permet de confirmer les résultats obtenus précédemment. En effet, dans le cadre de la résolution parallèle il est très difficile, à cause du non déterminisme, d'être catégorique lors de l'analyse des données expérimentales. Il est possible qu'avec de la « chance » un solveur soit considéré comme meilleur qu'un autre sans pour autant l'être réellement. Néanmoins, à la vue des résultats reportés dans cette section, nous pouvons affirmer que les gains observés dans les études expérimentales précédentes ne peuvent être imputées au hasard.

	SAT	MANYIDEM	MANYIDEM +	MANYSAT1.1	MANYSAT1.1+	MANYSAT1.5	MANYSAT1.5+
velev-live-uns-2.0-ebuf	N	8.2	**5.8**	7.1	**5.6**	**4.3**	4.4
velev-pipe-uns-1.0-8	N	825.1	**177.4**	472	**249.2**	55.7	**48.8**
velev-pipe-o-uns-1.1-6	N	114.3	**52.5**	78.9	**30.3**	9.5	9.5
manol-pipe-g10bidw	N	28.2	**20.9**	26.3	**19.6**	21.4	**19.8**
manol-pipe-c10nidw_s	N	10.6	**6**	9.5	**8.9**	14	**8.9**
manol-pipe-f10ni	N	**204.5**	204.9	88.5	**77**	122	**90.9**
manol-pipe-f9b	N	92.8	**67.1**	**37.8**	38.6	68.1	**50.7**
goldb-heqc-i10mul	N	168.4	**102.6**	46.2	**41.2**	**26**	29.9
goldb-heqc-term1mul	N	22.9	**19.2**	22.2	**21.2**	8.6	**5.8**
goldb-heqc-dalumul	N	383.9	**268**	582.1	**374.2**	**47.6**	58.6
mizh-sha0-35-3	Y	**5.2**	28.4	227	**38.6**	157.2	**5.5**
mizh-sha0-36-4	Y	–	–	–	**138.1**	–	**844.8**
gus-md5-06	N	9.3	**7.6**	9	**7.9**	**10.1**	10.9
gus-md5-05	N	3.1	**2.6**	2.9	**2.6**	3.6	**2.7**
gus-md5-09	N	361.6	**330.4**	376	**318.5**	376.9	**316.2**
partial-10-13-s	Y	–	**672.9**	779.5	**733**	–	–
partial-10-15-s	Y	–	**458.3**	**326.5**	548.4	–	**414.9**
dated-5-17-u	N	244.7	**160.6**	154.2	**125.1**	**139.7**	155.4
dated-5-15-u	N	146	**102.9**	**87.4**	122.9	102.5	**88.1**
uts-l06-ipc5-h33-unknown	N	12.4	**9.8**	15.9	**11.5**	15.7	**13.4**
sortnet-8-ipc5-h19-sat	Y	–	**837**	–	**880.2**	–	–
cube-11-h13-unsat	N	121.7	**68.2**	142.3	**91.2**	201.8	**104.4**
vliw_unsat_2.0/9dlx_vliw_at_b_iq1	N	179.2	**57.4**	76.6	**36**	10	**5.5**
vliw_unsat_2.0/9dlx_vliw_at_b_iq2	N	453.1	**153.9**	335.8	**145.7**	30	**21.3**
vliw_unsat_2.0/9dlx_vliw_at_b_iq3	N	–	**651.7**	840.9	**299**	96.9	**51.6**
vliw_unsat_2.0/9dlx_vliw_at_b_iq4	N	–	**649.3**	–	**393.9**	170.2	**128.8**
vliw_unsat_2.0/9dlx_vliw_at_b_iq5	N	–	–	–	**928.5**	331.9	**196.5**
vliw_unsat_2.0/9dlx_vliw_at_b_iq6	N	–	–	–	–	508.3	**349.8**
vliw_unsat_2.0/9dlx_vliw_at_b_iq7	N	–	**674.4**	–	–	803.4	**707.9**

TABLE 6.1 – Résultats obtenus par différentes versions du solveur MANYSAT intégrant ou non la stratégie d'ajustement dynamique de l'heuristique de choix de polarité. Les temps reportés dans ce tableau sont exprimés en secondes.

	SAT	MANYIDEM	MANYIDEM +	MANYSAT1.1	MANYSAT1.1+	MANYSAT1.5	MANYSAT1.5+
AProVE07-16	N	101	**78.7**	78.4	**74.3**	**65.3**	77.3
AProVE07-27	N	716.1	**529.7**	349	**288.1**	**482.6**	498.2
AProVE09-06	Y	15	**6**	16.9	**3.7**	269.5	**60.2**
AProVE09-20	Y	27.1	**12.2**	24.7	**13.7**	**11.1**	26
vmpc_29	Y	–	–	–	**153.7**	927.2	**464**
vmpc_34	Y	–	–	–	**142.1**	–	**405.4**
minandmaxor032	N	5.4	**2.7**	2.9	**2.8**	3	**2.7**
minxorminand032	N	7.9	**1.3**	2.5	**2.2**	4.8	**4.3**
minxorminand064	N	253.4	**12.3**	24.9	**22.5**	98.1	**89.4**
minxor128	N	–	**112**	78.9	**74.7**	**144**	146.3
maxxororand032	N	–	**588.7**	168.5	**142.1**	**83.7**	89.8
minxorminand128	N	–	**291.7**	350.1	**291.2**	–	–
q_query_3_L200_coli.sat	N	185	**145.3**	150.4	**94.1**	**104.2**	114.5
q_query_3_l45_lambda	N	109.8	**85.6**	80.9	**72.3**	**92**	115.1
q_query_3_L80_coli.sat	N	127.9	**44.7**	**28.5**	30.9	89	**62.4**
q_query_3_l44_lambda	N	108.1	**94.8**	98.9	**80.9**	94.6	**83.1**
q_query_3_L150_coli.sat	N	214.2	**148.7**	**137.5**	156.1	118.3	**98.5**
BioInstances/rpoc_xits_07	N	313.1	**236.1**	150.3	**131.1**	399.5	**283.4**
UR-10-5p0	N	14.2	**6.3**	8.5	**8.3**	8.8	**6.7**
UR-10-5p1	Y	4.8	**3.1**	3.1	**1.4**	6.1	**4**
UTI-20-10p0	N	–	**909.7**	833.9	**802.8**	**272**	328.2
ACG-20-10p0	N	–	**656.9**	–	**845.7**	–	**830.3**
UCG-20-10p1	Y	841.8	**673.4**	–	**651**	652.4	**553.2**
ACG-20-10p1	Y	–	–	–	–	920.8	**789.8**
UTI-20-10p1	Y	–	**784.8**	–	–	681.7	**638**
cmu-bmc-longmult15	N	5.6	**4.9**	5	**4.9**	7	**6.8**
post-cbmc-aes-ee-r2-noholes	N	311.6	**235.5**	**112.5**	128.7	211.8	**205.5**
post-cbmc-aes-d-r2-noholes	N	362.1	**212.2**	**122.6**	126.8	234.7	**203.2**
post-c32s-gcdm16-22	Y	79.4	**41.9**	50.7	**43.3**	28.5	**26.9**

TABLE 6.2 – Résultats obtenus par différentes versions du solveur MANYSAT intégrant ou non la stratégie d'ajustement dynamique de l'heuristique de choix de polarité. Les temps reportés dans ce tableau sont exprimés en secondes.

6.4 Conclusion

Des travaux basés sur l'utilisation de l'heuristique de polarité pour régler certaines composantes des solveurs SAT modernes ont déjà été menés par le passé. Ainsi, dans Biere (2008a), une politique de redémarrage basée sur l'heuristique de polarité *progress saving* a été proposée. Dans ce chapitre, nous avons présenté une nouvelle mesure basée sur ce principe permettant d'estimer la distance entre deux unités de calcul. Cette mesure a ensuite été utilisée dans le cadre d'un solveur parallèle de type *portfolio* en ajustant dynamiquement l'heuristique de polarité des différentes unités de calcul afin de diversifier la recherche. Les résultats expérimentaux, menés sur l'ensemble des instances industrielles de la compétition SAT 2009, ont montré que notre méthode améliore sensiblement les performances du solveur MANYSAT 1.1. En ce qui concerne l'ajout de notre approche dans le solveur MANYSAT 1.5, nous ne pouvons pas conclure qu'il y a un réel gain. Puisque le solveur MANYSAT 1.5 est un solveur ayant une structure qui effectue un compromis entre la diversification et l'intensification Guo *et al.* (2010). Néanmoins, à la vue des résultats (sur les instances industrielles) il semble que notre approche soit très prometteuse.

Plusieurs perspectives s'ouvrent naturellement à la vue de ces résultats. Tout d'abord, nous envisageons d'étudier le comportement de notre approche sur les autres familles d'instances (instances aléatoires et académiques). De plus, il semble que la fréquence selon laquelle l'ajustement est effectué influe sur les performances. Il semble donc nécessaire d'étudier plus finement ce phénomène et de définir une stratégie dynamique pour régler ce paramètre. Nous pensons par la suite étudier d'autres critères d'ajustement. En effet, au lieu de se limiter à inverser l'heuristique de choix de polarité comme nous l'avons fait, une autre approche peut consister à en changer complètement pendant la phase de recherche. De cette manière un *portfolio* d'heuristiques de polarités est envisageable. Une autre perspective consiste à utiliser les informations fournies par la notion d'intention pour contrôler les flux de clauses apprises transitant entre les différents cœurs d'un solveur parallèle.

Pour terminer, nous envisageons d'étudier notre approche sur d'autres solveurs de type *portfolio* (PLINGELING Biere (2010), CRYPTOMINISAT// Soos (2011), *etc.*) et plus particulièrement sur le solveur parallèle *portfolio* déterministe proposé par Hamadi *et al.* (2011). En effet, considérer une telle approche permet de simplifier fortement le protocole expérimental et surtout de supprimer le critère « chance » intrinsèque à l'utilisation de telles méthodes.

Chapitre 7

Stratégies d'élimination des clauses apprises

Sommaire

L'APPRENTISSAGE de clauses est un composant important des solveurs SAT modernes. Toutefois le nombre de clauses apprises peut être exponentiel dans le pire des cas. Pour gérer ces clauses, plusieurs politiques de nettoyage ont été proposées basées principalement sur la suppression de clauses jugées non pertinentes pour la suite de la recherche. Dans ce chapitre nous donnons, tout d'abord, une comparaison des différentes stratégies proposées dans la littérature avec une stratégie simple basée sur un choix aléatoire des clauses à supprimer. Nous proposons ensuite de nouveaux critères permettant de juger de la pertinence d'une clause par rapport à un état de la recherche. Nous abordons ensuite, la question de la fréquence de nettoyage afin de gérer le temps de vie d'une clause apprise dans une stratégie de nettoyage. Dans la section expérimentations, nous comparons d'abord les performances des différentes stratégies (des combinaisons de critères et des différentes fréquences définie précédemment), et nous sélectionnons ensuite les stratégies de réduction complémentaires à intégrer dans un solveur parallèle de type portfolio.

7.1 Introduction

Au cours de ces deux dernières décennies le problème SAT a énormément gagné en intérêt, avec l'arrivée d'une nouvelle génération de solveurs. Ces solveurs, appelés solveurs CDCL (Conflict Driven, Clause Learning) Moskewicz *et al.* (2001), Eén et Sörenson (2004), ont pour la première fois été capables de résoudre des instances conséquentes codant des problèmes de la vie réelle. Leur architecture est basée sur les composants suivants : des heuristique de choix de variables VSIDS (Variable State Independant,

Decading Sum) Moskewicz *et al.* (2001), des redémarrages Huang (2007), Gomes *et al.* (1998), et des structures de données efficaces (ex. Watched literals) ainsi que l'apprentissage de clauses Marques-Silva et Sakallah (1996b), Moskewicz *et al.* (2001), Zhang *et al.* (2001). D'un point de vue théorique, Knot Pipatsrisawat et Adnan Darwiche Pipatsrisawat et Darwiche (2009) ont récemment prouvé que l'apprentissage de clauses tel qu'il est fait dans les démonstrateurs SAT modernes permet de simuler la résolution générale. Ce résultat montre que l'apprentissage de clauses rend les démonstrateurs de type DPLLDavis *et al.* (1962) aussi puissants que la résolution générale. Mais en pratique, cet apprentissage peut induire certains problèmes. En effet, le nombre de clauses apprises est connu pour être exponentiel dans le pire des cas. Dès lors de point de vue technique, il faut trouver des moyens efficaces de gérer ce gros volume de clauses. Pour maintenir une base des clauses apprises de taille polynomiale permettant aussi un coût raisonnable pour la propagation unitaire, un ensemble de stratégies a été proposé pour supprimer périodiquement les clauses supposées inutiles pour la suite de la recherche. Parmi ces stratégies, deux ont montré des résultats pratiques intéressants. La première nommée CSIDS (*Clause State Independent, Decaying Sum*), la plus populaire, est basée sur la notion de *first fail*. Cette stratégie considère que les clauses ne participant pas aux conflits récents devraient être supprimées. La seconde est basée sur une mesure nommée LBD (*Literal Block Distance*)Audemard et Simon (2009a). Cette mesure correspondant au nombre de niveaux différents intervenant dans la génération de la clause apprise. Les auteurs considèrent alors que les clauses ayant un LBD élevé sont inutiles pour la suite de la recherche et qu'elle peuvent donc être supprimées. Plus récemment, une nouvelle heuristique dynamique pour la gestion de la base de clauses apprises est proposée dans Audemard *et al.* (2011). Cette approche s'appuie sur le principe d'activation et de désactivation de clauses. Étant donnée une étape de la recherche, elle utilise une fonction pour activer certaines clauses supposées pertinentes et pour en désactiver d'autres. Le maintien d'une base pertinente de taille polynomiale est crucial pour l'efficacité de l'apprentissage dans les solveur SAT. En effet, conserver un grand nombre de clauses peut avoir des conséquences néfastes sur l'efficacité de la propagation unitaire.

Dans ce chapitre nous donnons, tout d'abord, une comparaison des différentes stratégies proposées dans la littérature avec une stratégie simple basée sur un choix aléatoire des clauses à supprimer. Nous proposons ensuite de nouveaux critères permettant de juger de la pertinence d'une clause par rapport à un état de la recherche. Ces critères seront utilisés ensuite avec des différentes fréquences de réduction pour définir les différentes façons de gérer la base des clauses apprises. Avant de conclure et donner quelques perspectives, nous donnons les résultats expérimentaux obtenus sur les instances de la compétition SAT 2011.

7.2 Motivation

Les solveurs CDCL, disposent d'une fonction appelée *reduceDB*. Cette fonction, bien que souvent omise de la description des démonstrateurs CDCL, est pourtant cruciale pour leurs performances. En effet, conserver un grand nombre de clauses peut altérer l'efficacité de la propagation unitaire, tandis que supprimer trop de clauses peut rendre l'apprentissage inefficace. Par conséquent, identifier les bonnes clauses apprises (*i.e.* importantes pour la dérivation de la preuve) est un véritable challenge. La première mesure proposée pour qualifier la qualité d'une clause apprise CSIDS utilise l'idée sous-jacente à l'heuristique VSIDS. Plus précisément, une clause apprise est considérée comme utile pour la preuve, si elle apparaît fréquemment impliqué dans les conflits récents. Une telle stratégie suppose qu'une clause utile dans le passé le sera dans le futur. Plus récemment, une nouvelle mesure, nommée LBD, a été utilisée pour estimer la qualité d'une clause apprise. L'implémentation de cette mesure dans un solveur CDCL (le solveur GLUCOSE Audemard et Simon (2009a)) a permis d'en améliorer sensiblement les

performances. Cette nouvelle mesure est basée sur le calcul du nombre de niveaux de décision apparaissant dans une clause apprise lors de sa génération. Les auteurs ont démontré expérimentalement que les clauses possédant un faible LBD sont plus souvent utilisées pendant la recherche que les clauses ayant un LBD élevé. Plus récemment, les auteurs de Audemard *et al.* (2011) ont proposé une heuristique dynamique de gestion de la base de clauses apprises, ils ont utilisé un critère nommé PSM (Progress Saving based Measure), les clauses apprises avec une petite valeur de PSM ont de fortes chances d'être utilisées dans la propagation unitaire ou d'être falsifiées. Au contraire, une clause avec un grand PSM a plus de chance d'être satisfaite par plus d'un littéral et donc d'être inutile pour la suite de la recherche.

Algorithme 7.1 : Stratégie de réduction

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

```
si faireReduction() alors
    TrierLesClausesApprises;        /* Selon les critères définis */
    limit = n/2;
    ind = 0;
    tant que ind < limit faire
        clause = Δ[ind] ;
        si clause.size() > 2 alors
            EliminerLaCLause;
        sinon
            GarderLaClause;
        ind++;
retourner Δ ;
```

L'algorithme 7.1 présente le schéma d'une stratégie de réduction.

Notons bien que la stratégie de suppression des clauses doit indiquer la fréquence des suppressions, le nombre et le type de clauses à supprimer. Dans ce chapitre, nous nous focalisons sur l'identification des clauses apprises pertinentes et l'évaluation des performances en variant la fréquence de réduction. Nous garderons le même nombre de clauses à supprimer que le solveur de référence.

Pour avoir un point de vue global sur les différentes stratégies de gestion de la base de clauses apprises et leurs efficacités, nous menons une première expérimentation sur les instances de la dernière compétition internationale SAT 2011. Nous comparons les deux stratégies de suppression de clauses apprises les plus connues CSIDS et LBD avec une stratégie simple de type RANDOM qui attribue une activité aléatoire aux clauses. Les trois stratégies sont intégrées dans le solveur Minisat2.2. De façon similaire aux approches de réduction classique, l'ensemble des clauses apprises est tout d'abord trié par ordre croissant (*r*esp. décroissant) selon leurs activités (*r*esp. LBD, PSM). Une fois les clauses triées la base de clauses apprises est réduite de moitié. Comme pour les autres stratégies, nous conservons toujours les clauses binaires, et nous n'avons pas changé la fréquence de nettoyage de la base. Toutes nos expérimentations (pour celles dans la suite aussi) ont été menées sur un cluster Quad-Core Intel XEON X5550 avec 32Gb de mémoire. 300 instances ont été utilisées, elles sont issues de la compétition SAT 2011, catégorie industrielle. Pour l'ensemble des expérimentations le temps CPU est limité à 1 heure.

Pour chaque solveur, nous indiquons le nombre d'instances résolues (#Résolues) ainsi que le nombre d'instances résolues satisfiables (#SAT) et insatisfiables (#UNSAT). Nous donnons aussi la moyenne du temps nécessaire pour résoudre une instance (tps moy). Le temps moyen est calculé sur l'ensemble des 300 instances, 3600 secondes est utilisé pour chaque instance non résolue.

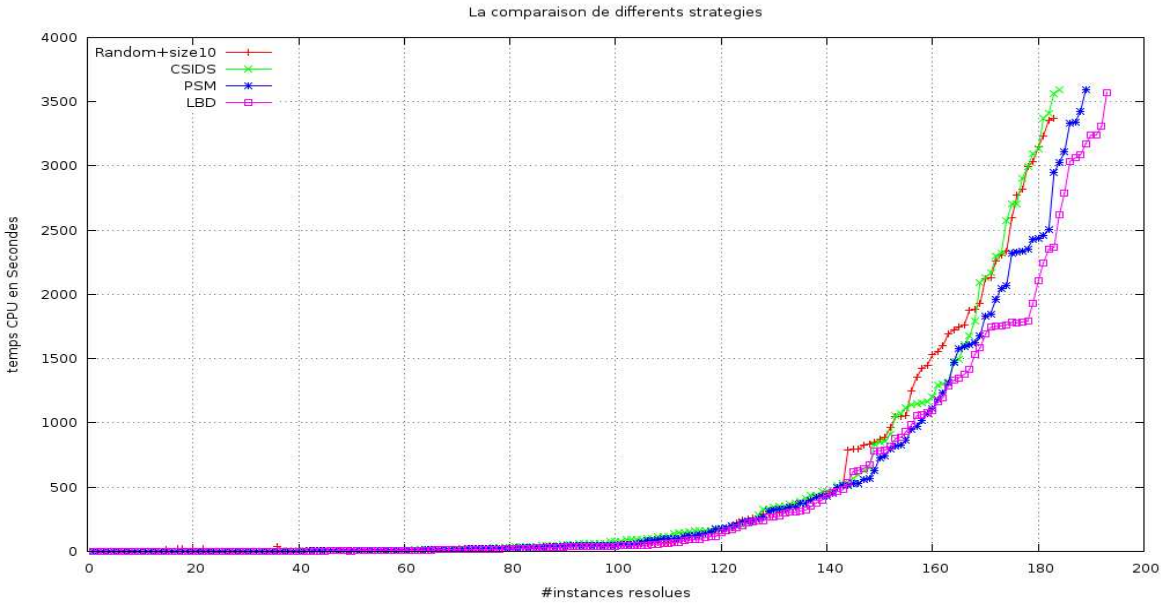


FIGURE 7.1 – Nombre d'instances résolues en fonction du temps pour le solveur Minisat2.2 intégrant les différentes stratégies de l'état-de-l'art et la stratégie RANDOM.

	Minisat2.2	
Stratégies	#Résolues (#SAT-#UNSAT)	tps moy
RANDOM	182 (**92**-90)	1670.12s
CSIDS	184 (88-96)	1673.83s
LBD	193 (**92**-101)	**1596.66s**
PSM	189 (88-101)	1628.84s

TABLE 7.1 – Comparaison des stratégies de état-de-l'art avec une stratégie RANDOM intégrée dans MiniSAT2.2

La Figure 7.1 montre les résultats obtenus dans la limite du temps autorisé.

Les Tableaux 7.1 détaillent les résultats obtenus. On peut aisément constater que la stratégie LBD est la meilleure. La surprise provient du résultat de la stratégie simple RANDOM. En effet, contrairement à notre prédiction, la stratégie RANDOM apparaît compétitive en résolvant seulement 2 instance de moins que CSIDS et 7 instances de moins que PSM. De plus, cette stratégie simple, a résolue le plus d'instances satisfiables (92). Cette observation soulève la question de l'efficacité et de la pertinence des stratégies proposées dans la littérature. Ceci nous conforte dans notre objectif de chercher d'autres critères plus pertinents.

Dans la section suivante, nous présentons de nouveaux critères qui permettent d'évaluer la pertinence des clauses apprises à partir d'un état de recherche.

7.3 Nouveaux critères d'identification de l'importance d'une clause apprise

Comme précisé précédemment, les solveurs SAT de type CDCL peuvent être reformulés comme un système de preuve par résolution Pipatsrisawat et Darwiche (2009), Beame *et al.* (2004). Le principal inconvénient des méthodes basées sur la résolution générale est lié à leur complexité en espace. D'un certain point de vue, les solveurs SAT modernes peuvent être vus comme une méthode de preuve par résolution avec une stratégie de suppression de clauses. Par conséquence, la complétude des solveur SAT modernes est fortement liée à la politique de suppression de clauses et aux redémarrages. Par exemple, supposons que la stratégie de nettoyage de la base de clauses apprises soit très agressive, dans ce cas il nous serait impossible de garantir la complétude du solveur. Définir quelle clause serait utile pour la preuve est une question importante pour l'efficacité des solveurs. Répondre à une telle question est cependant difficile d'un point de vue calculatoire et est très proche du problème qui consiste à trouver une preuve de taille minimale. Pour apporter une solution à ce problème nous définissons des critères permettant d'évaluer la pertinence d'une clause apprise et nous évaluons expérimentalement leurs efficacités.

7.3.1 Représentation d'une clause apprise

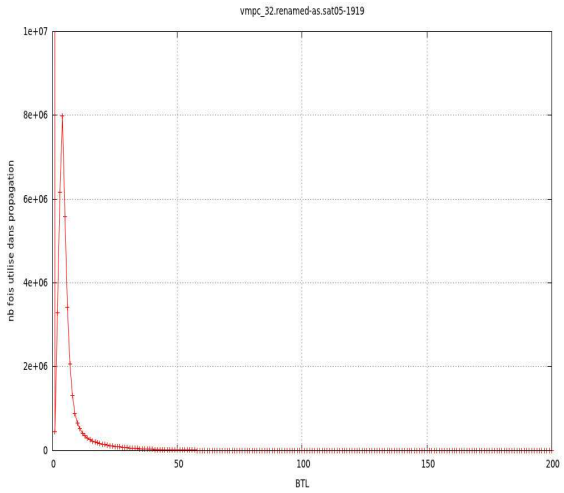
Une clause apprise peut être représentée comme suit : $c = x_1^{i_1} \vee x_2^{i_2} \vee x_3^{i_3} \vee \ldots x_n^{i_n}$ où $x_j^{i_j}$ signifie que le littéral x_j est falsifié au niveau i_j. Il faut noter que les niveaux des littéraux d'une même clause varient au cours de la recherche. En utilisant cette information dynamique, nous allons définir deux critères, basés sur les niveaux des littéraux dans une clause apprise, pour estimer la pertinence de cette clause.

7.3.2 L'activité basé sur le niveau du saut arrière

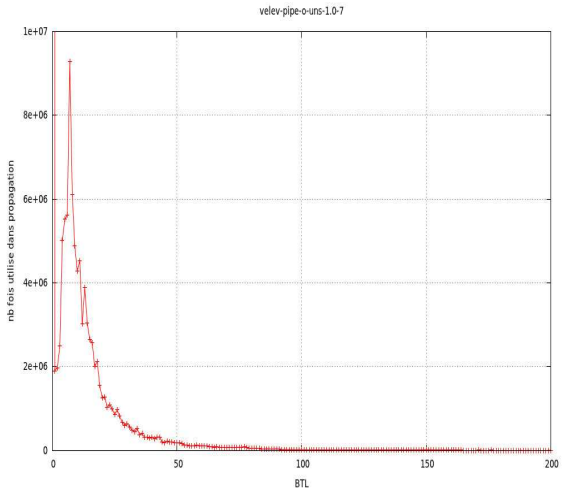
Chaque fois qu'une clause apprise est générée, elle permet au solveur d'effectuer un retour en arrière à un certain niveau de l'arbre de recherche. Dans notre approche, le niveau de saut arrière : BTL (*B*ackTrack Level) est utilisé pour estimer l'importance d'une clause apprise. Une clause est considérée pertinente si elle permet au solveur d'effectuer un saut en arrière àu niveau le plus haut possible (le plus proche de la racine). L'intuition est basée sur les faits suivants. D'abord, les littéraux de décision choisis en haut de l'arbre de recherche correspondent aux variables les plus pondérés selon VSIDS et donc les plus importantes. Deuxièmement, chaque clause apprise est considérée pertinente si elle nous permet de couper des branches en haut de l'arbre de recherche. En effet, plus haut est le niveau de backtrack, plus la clause apprise partage des variables avec la partie la plus haute de la branche, et donc plus elle sera utilisée pour couper l'espace de recherche.

Notre critère est défini comme suit : pour chaque clause apprise c, au moment de sa génération, nous lui associons une valeur $BTL(c)$, le niveau du saut arrière. Pour toutes les clauses apprises, cette valeur de BTL peut être mise-à-jour comme suit : si une clause apprise c est utilisée pour propager un littéral au niveau i, et $i < BTL(c)$, alors nous adaptons $BTL(c) = i$. De cette façon, nous gardons son niveau le plus haut, où la clause a été la raison d'un littéral propagé.

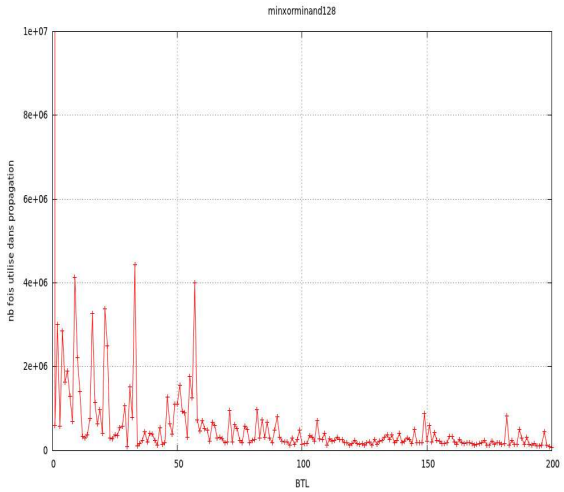
Pour analyser et valider cette hypothèse, un ensemble d'expérimentations ont été menées. La Figure 7.2 reporte, pour quelques instances représentatives, le nombre de fois où une clause avec une certaine valeur de BTL a été utilisée durant le processus de propagation unitaire. Dans cette expérimentation, lorsqu'une clause c provenant de la base de clauses apprises est utilisée dans le processus de propagation



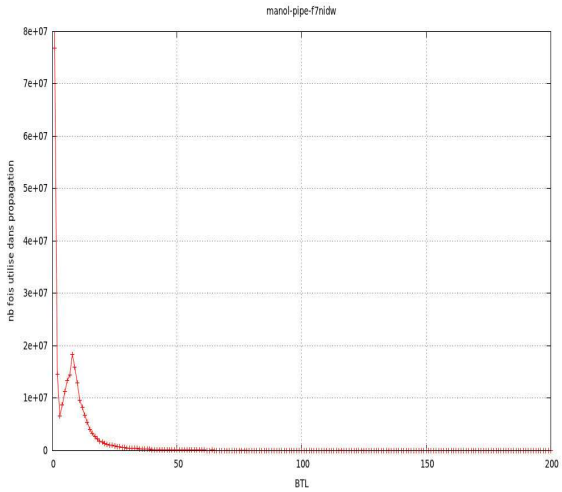
(a) vmpc-32renamed-as.sat05-1919



(b) velev-pipe-o-uns-1.0-7



(c) minxorminand128



(d) manol-pipe-f7nidw

FIGURE 7.2 – Nombre de fois qu'une clause est utilisée par PU en fonction de BTL

unitaire nous incrémentons $\alpha(BTL)$ (qui correspond au nombre de fois où une clause avec une telle valeur de BTL est utilisée pour propager un littéral).

D'après la figure 7.2, les clauses apprises possédant une petite valeur de BTL (axe des abscisses) sont plus souvent utilisées dans le processus de propagation (axe des ordonnés) que les clauses possédant une grande valeur de BTL. En analysant plus en détail ces données, nous pouvons constater que les clauses les plus utilisées ont une valeur de BTL inférieur à 10. Nous pouvons donc affirmer, puisque

l'expérimentation a été menée sur un très large panel d'instances, que pour la majorité des instances considérées la distribution de la valeur de BTL ressemble à celle des deux courbes en haut de la figure 7.2.

7.3.3 La distance basée sur les niveaux

Pour une clause apprise $c = x_1^{i_1} \vee x_2^{i_2} \vee x_3^{i_3} \vee \ldots x_n^{i_n}$, elle a deux littéraux $x_j^{i_j}$ et $x_k^{i_k}$ qui sont de niveau plus haut (respectivement plus bas) dans l'arbre de recherche, telle que $\forall x_m^{i_m} \in c, i_m \geq i_j$ et $\forall x_m^{i_m} \in c, i_m \leq i_k$. Nous définissons la distance d'une clause apprise comme suit : $Dist(c) = i_k - i_j$; où i_k et i_j sont respectivement les niveaux plus bas et plus haut d'un littéral dans la clause c. Ce critère est intéressant, supposons qu'on a une clause $c = x_1^{45} \vee x_2^{45} \vee x_3^{45}$, $BTL(c) = 45$ et $Dist(c) = 0$. Selon le critère BTL, cette clause est considérée peu importante puisqu'elle a une valeur BTL grande. Cependant si on affectait $\neg x_1$ en haut de l'arbre de recherche par exemple au niveau 1, cette clause c a de fortes chances d'être utilisée pour propager un littéral au niveau 1, donc jugée importante dans ce cas là. Cette différence entre le niveaux max et min d'une clause, évalue l'étendue des littéraux de la clauses sur la branche courante.

Si on considère que le critère BTL qui représente le pouvoir actuel de couper la branche de l'arbre de recherche d'une clause apprise, alors le critère DISTANCE signifie le pouvoir potentiel de cette clause.

On peut également constater que pour chaque clause apprise c on a toujours $LBD(c) \leq Dist(c) + 1$.

7.4 Fréquences d'élimination

Comme nous l'avons présenté dans 3.5, une stratégie d'élimination doit répondre aux trois questions suivantes :

- Quelle type de clauses à supprimer ou à conserver ?
- Avec quelle fréquence doit-on réduire la base des clauses apprises ?
- Combien de clauses à supprimer ?

Nous avons déjà défini deux nouveaux critères pour identifier les clauses apprises pertinentes. Nous allons maintenant analyser l'impact de la fréquence de réduction de la base des clauses apprises sur les performances des solveurs SAT.

La fréquence de réduction est importante, puisqu'elle a une influence directe sur la durée de vie d'une clause apprise. On peut noter que l'importance d'une clause peut évoluer au cours de la recherche. Une clause apprise est par exemple "1- empowerment" au moment de sa génération, alors qu'elle peut ne plus l'être après la génération d'autres clauses apprises Pipatsrisawat et Darwiche (2009). Une clause actuellement considérée comme peu importante pourrait donc être nécessaire par la suite ou inversement. En général, une réduction agressive permet de garder une base de clauses apprises plus petite et maintenir une vitesse du propagation unitaire, mais elle risque aussi d'enlever des clauses apprises nécessaires. En revanche, une réduction prudente ne réduisant pas souvent la base de clauses apprises afin d'éviter de supprimer des clauses pertinentes pourra avoir pour effet de ralentir le processus de propagation unitaire et donc de dégrader les performances du solveur.

Dans cette section, nous allons présenter trois différentes fréquences afin de gérer l'intervalle entre deux réduction.

7.4.1 Fréquence de Minisat - FréqMiniSAT

La première approche que nous avons choisi est celle implémentée dans MINISAT Eén et Sörenson (2004) qui est le solveur le plus utilisé actuellement. Sa fonction de réduction de la base des clauses apprises est appelée une fois que la taille de celle-ci dépasse un certain un seuil, ce seuil est initialisé à $1/3$ de la taille de la base de clauses initiale. Ce seuil est augmenté de 10% lorsque le nombre de clauses apprises est supérieur à 100×1.5^n où n représente le nombre d'augmentations. Les différentes constantes sont fixées expérimentalement.

7.4.2 Fréquence de Glucose - FréqGlucose

La deuxième fréquence que nous avons choisi est la stratégie décrite dans le solveur GLUCOSE Audemard et Simon (2009a). Elle consiste à appeler la fonction de réduction de la base de clauses apprises lorsque le nombre de conflits obtenu depuis le dernier appel est supérieur à $4000 + 300 \times n$, n représente le nombre d'appels à la fonction de réduction.

Avec cette fréquence, la fonction de réduction de la base de clauses apprises sera appelée plus souvent que celui avec la fréquence 1.

7.4.3 Une fréquence prudente - FréqPrudent

Nous avons présenté une fréquence classique (la fréquence de Minisat) et une fréquence qui réduit la base de clauses apprises plus souvent (la fréquence de Glucose). Maintenant, nous allons définir une fréquence prudente . Nous voulons qu'avec cette fréquence, la fonction de réduction de la base de clauses apprises soit appelée moins souvent, les clauses apprises seront gardées plus longtemps qu'auparavant. Donc, une suite géométrique est définie. La fonction de réduction est appelée lorsque le nombre de nouveaux conflits est supérieur à 4000×1.2^n, n représente le nombre de réductions.

Avec cette fréquence de réduction, la durée de vie d'une clause apprise est plus longue.

Dans la section suivante, nous présentons une comparaison des performances de ces trois fréquences de réduction.

7.5 Expérimentations

Dans cette section nous allons d'abord comparer, du coté séquentiel, l'évaluation des performances de nos nouveaux critères avec celles de l'état de l'art en les combinant avec les différentes fréquences de réduction présentées auparavant. Ensuite, en analysant la complémentarité de ces stratégies, nous allons en sélectionner 4 et les intégrer dans le solveur parallèle MANYSAT.

Notons que pour toutes les stratégie dans cette section, une moitié des clauses apprises de la base sera supprimée à chaque réduction.

7.5.1 Intégration dans MINISAT

Comparaison avec la fréquence de MINISAT

Nous comparons d'abord nos critères BTL et DISTANCE et les 3 critères de l'état de l'art implémentés dans MINISAT avec sa propre fréquence de réduction. La seule différence ici est le critère de suppression.

	Minisat2.2-F1	
Stratégies	#Résolues (#SAT-#UNSAT)	tps moy
CSIDS	184 (88-96)	1673.83s
LBD	193 (92-101)	1596.66s
PSM	189 (88-101)	1628.84s
BTL	196 (90- **106**)	1566.56s
DISTANCE	**197** (**93**-104)	**1561.94s**

TABLE 7.2 – Comparaison des critères de l'état-de-l'art intégrés dans MiniSAT2.2 avec FréqMiniSAT

Le tableau 7.2 montre les résultats obtenus. Pour chaque critère, nous indiquons le nombre d'instances résolues (#Résolues) ainsi que le nombre d'instances résolues satisfiables (#SAT) et insatisfiables (#UNSAT). Nous donnons aussi la moyenne du temps nécessaire pour résoudre une instance (tps moy). Le temps moyen est calculé sur l'ensemble des 300 instances, 3600 secondes sont utilisées si l'instance n'est pas résolue en moins d'une heure.

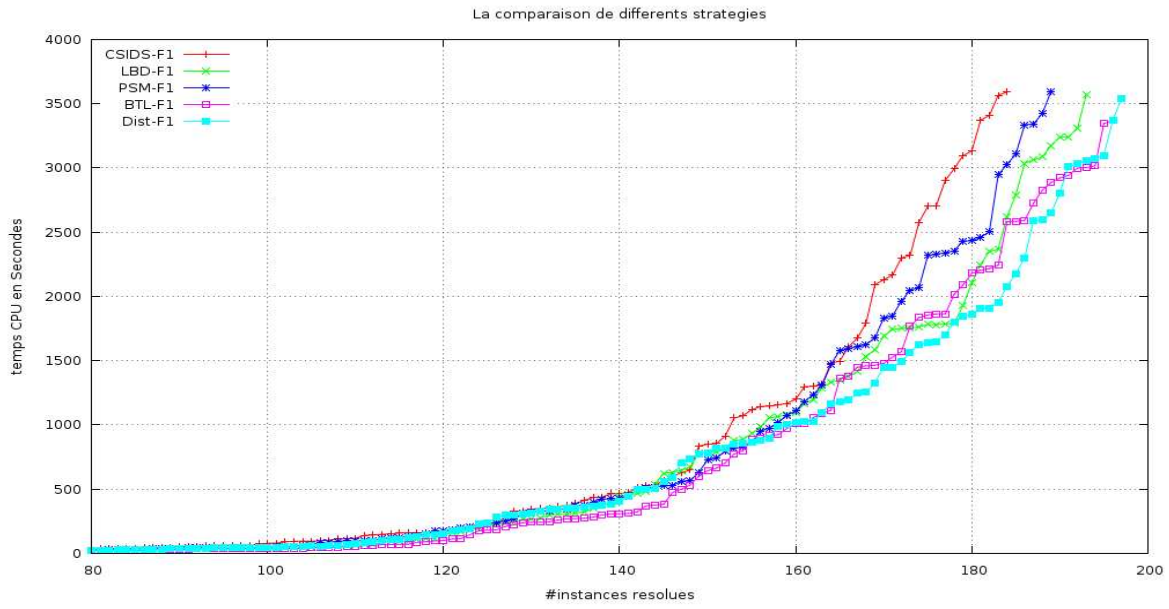


FIGURE 7.3 – Comparaison avec les critères de l'état-de-l'art intégrés dans MiniSAT2.2 avec FréqMiniSAT

La Figure 7.3 contient les courbes permettant de comparer le nombre d'instances résolues en fonction du temps pour les cinq critères.

Il est claire que nos critères BTL et DISTANCE sont nettement meilleurs que les trois critères de l'état de l'art. Avec le critère DISTANCE, le solveur MINISAT résout 197 instances (93 SAT et 104 UNSAT), ce qui est meilleur que les autres critères, il est aussi le plus rapide des cinq. L'autre critère BTL résout seulement une instance de moins, est classé deuxième et il a résolu le plus d'instance UNSAT.

Instance	Statues	CSIDS	LBD	PSM	BTL	DISTANCE
1dlx_c_iq60_a	UNSAT	-	3035.65s	2429.80s	2723.88s	2590.26s
aes_32_3_keyfind_1	SAT	2130.30s	-	1963.30s	-	**1449.90s**
aes_64_1_keyfind_1	SAT	-	1288.76s	-	**799.09s**	1564.16s
AProVE07-01	UNSAT	-	-	-	**2825.70s**	-
AProVE11-06	UNSAT	-	**2791.03s**	-	3370.17s	3540.33s
clauses-8.renamed-as.sat05-1964	SAT	-	1780.53s	2066.48s	1766.75s	**1625.50s**
comb1.shuffled	UNSAT	848.69s	-	2317.59s	**1864.37s**	2592.90s
dated-10-17-u	UNSAT	-	-	**3423.48s**	-	-
eq.atree.braun.11.unsat	UNSAT	-	3083.93	2048.45s	**1458.61s**	3013.13s
gss-22-s100.cnf	SAT	3095.01s	-	**629.62s**	2583.88s	1908.97s
gss-27-s100	SAT	-	-	-	-	**1192.10s**
homer16.shuffled	UNSAT	-	3308.22s	-	**2886.15s**	3071.66s
hwmcc10-timeframe-expansion-k45-pdtvissoap1-tseitin	UNSAT	3565.70	3572.95s	-	3344.66s	**3098.14s**
ibm-2002-30r-k85	SAT	2576.60s	878.50s	**564.26s**	-	-
k2mul.miter.shuffled-as.sat03-355	UNSAT	-	1792.10s	1848.91s	1360.50s	**1327.10s**
minandmaxor128	UNSAT	2905.08s	-	**2503.78s**	3015.67s	3054.23s
myciel6-tr.used-as.sat04-320	UNSAT	-	-	-	**2087.45s**	-
ndhf_xits_19_UNKNOWN	SAT	**853.62s**	-	-	-	881.75s
partial-10-15-s	SAT	-	3241.60s	-	**2942.34s**	3030.63s
rbcl_xits_08_UNSAT	UNSAT	-	-	-	**2998.96s**	3368.38s
SAT_dat.k100	UNSAT	-	1750.18s	3027.93s	2197.12s	**1177.55s**
SAT_dat.k85	SAT	2318.61s	819.24s	**527.44s**	-	-
sha0_36_5	SAT	-	263.98s	97.29s	**25.61s**	1019.84s
slp-synthesis-aes-bottom13	UNSAT	**1300.07s**	1582.12s	1580.53s	-	2077.62s
slp-synthesis-aes-bottom14	UNSAT	-	-	**2434.34s**	-	-
slp-synthesis-aes-top25	SAT	-	-	-	2216.92s	**1857.59s**
slp-synthesis-aes-top26	SAT	-	1166.29s	3594.68s	245.80s	**186.58s**
smtlib-qfbv-aigs-VS3-benchmark-S2-tseitin	UNSAT	-	-	-	**3002.81s**	-
smtlib-qfbv-aigs-servers_slapd_a_vc149789-tseitin	SAT	-	**1532.45s**	2945.79s	-	2298.02s
sortnet-8-ipc5-h19-sat	SAT	**1142.01s**	3240.23s	-	1479.28s	-
vmpc_34.renamed-as.sat05-1926	SAT	-	**92.57s**	-	-	-
x1mul.miter.shuffled-as.sat03-359	UNSAT	-	**1780.10s**	2338.86s	2202.49s	1848.80s

TABLE 7.3 – 2011 SAT Compétition, Industriel : temps (s) résultats 5 critères avec FréqMiniSAT

Le tableau 7.3 liste les instances complémentaires pour les cinq critères, les instances qu'au moins 1 critère n'a pas résolu en une heure. Nous pouvons constater que même nos deux critères sont meilleurs sur le plupart des instances, il n'y a pas de critère qui domine tous les autres sur l'ensemble des instances.

Cette constatation nous inspire l'idée de différencier les bases de clauses apprises dans un solveur parallèle de type portfolio.

Comparaison avec la fréquence de GLUCOSE

Pareillement, une comparaison avec FreqGlucose est présentée dans cette section.

	Minisat2.2-F2	
Stratégies	#Résolues (#SAT-#UNSAT)	tps moy
CSIDS	189 (**96**-93)	1605.96s
LBD	189 (92-97)	1588.82s
PSM	188 (89-99)	1594.60s
BTL	189 (91-98)	1584.00s
DISTANCE	190 (92-98)	1550.03s

TABLE 7.4 – Comparaison des critères de l'état-de-l'art intégrés dans MiniSAT2.2 avec FréqGlucose

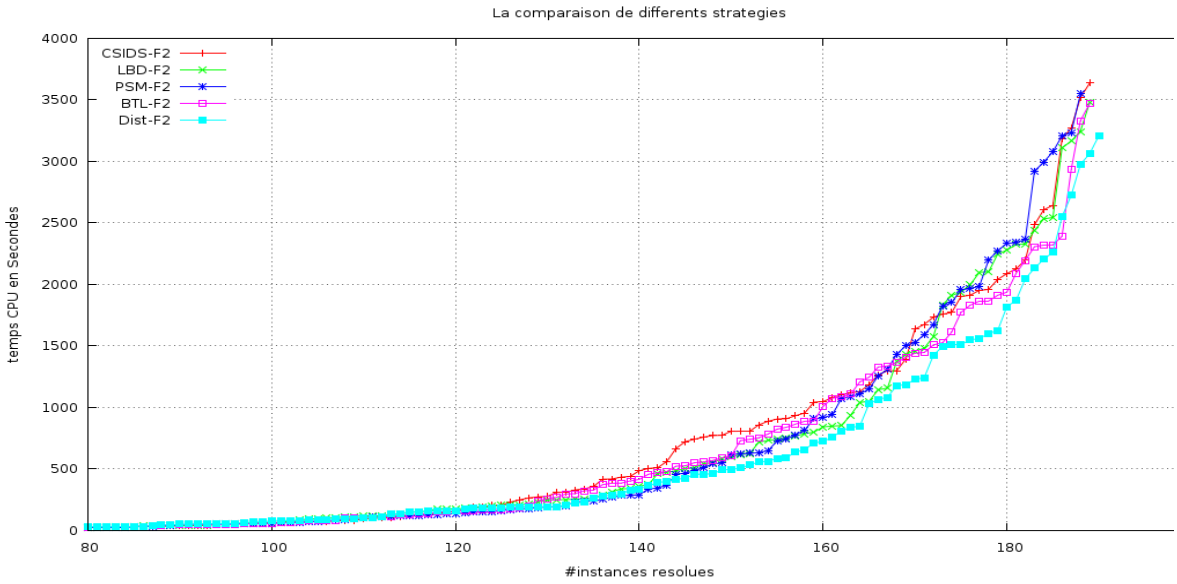


FIGURE 7.4 – Comparaison des critères de l'état-de-l'art intégrés dans MiniSAT2.2 avec FréqGlucose

Instance	Statues	CSIDS	LBD	PSM	BTL	DISTANCE
aaai10-planning-ipc5-pathways-17-step21	SAT	2126.04	**55.02**	-	-	206.05
aes-32-5-keyfind-1	SAT	**1129.55**	-	-	-	-
aes-64-1-keyfind-1	SAT	2606.06	2327.37	-	-	-
AProVE07-01	UNSAT	-	-	3206.55	**2933.43**	-
blocks-blocks-36-0.120-NOTKNOWN	UNSAT	-	3478.98	-	-	**2550.87**
blocks-blocks-37-1.130-NOTKNOWN	UNSAT	3517.98	1910.20	-	-	**1496.18**
cube-11-h14-sat	SAT	1757.42	-	-	782.77	**590.40**
dated-10-17-u	UNSAT	-	-	**1825.41**	-	-
dated-5-13-u.cnf	UNSAT	-	-	**2332.84**	-	-
gripper13u.shuffled-as.sat03-395	UNSAT	-	1575.17	-	1244.71	**710.96**
gss-21-s100	SAT	3271.08	-	3077.73	2320.47	**506.91**
gss-22-s100	SAT	**69.73**	472.93	-	1107.17	3207.63
homer16.shuffled	UNSAT	-	-	-	-	**2260.10**
hwmcc10-timeframe-expansion-k45-pdtpmsgoodbakery-tseitin	UNSAT	-	1995.61	-	2391.16	**1064.98**
hwmcc10-timeframe-expansion-k45-pdtvissoap1-tseitin	UNSAT	-	3243.20	3233.72	**1612.17**	-
k2fix-gr-rcs-w8.shuffled	UNSAT	**3180.65**	-	-	-	-
k2mul.miter.shuffled-as.sat03-355	UNSAT	-	2100.93	1956.63	**1771.42**	1869.98
manol-pipe-f7idw	UNSAT	-	95.47	**61.26**	295.92	336.50
ndhf-xits-19-UNKNOWN	SAT	**6.43**	-	-	-	-
partial-10-13-s	SAT	**1771.31**	-	2196.03	1830.95	-
rbcl-xits-08-UNSAT	UNSAT	3635.23	-	2365.47	**1862.45**	-
slp-synthesis-aes-bottom14	UNSAT	-	-	**2920.47**	-	-
slp-synthesis-aes-top25	SAT	2197.26	-	**509.53**	-	-
slp-synthesis-aes-top26	SAT	1117.29	2247.04	-	**739.55**	1818.63
smtlib-qfbv-aigs-servers-slapd-a-vc149789-tseitin	SAT	-	2324.69	-	**1336.71**	-
total-10-17-u.cnf	UNSAT	-	-	2338.94	-	-
UTI-20-10p0	SAT	1733.27	1833.82	-	2194.24	**1029.27**
UTI-20-10p1	SAT	**2641.94**	3109.10	2992.80	-	2722.77
valves-gates-1-k617-unsat	UNSAT	**2484.23**	-	3547.32	3330.46	3061.67
vmpc-35.renamed-as.sat05-1921	SATISFIABLE	-	**1039.86**	-	-	1081.87
x1mul.miter.shuffled-as.sat03-359	UNSAT	-	**1457.99**	1673.87	2318.90	1559.67

TABLE 7.5 – 2011 SAT Compétition, Industriel : temps (s) résultats 5 critères avec FréqGlucose

Selon le tableau 7.4 et la figure 7.4, on peut voir que les cinq critères, avec une fréquence assez agressive, ne présentent pas beaucoup de différence ni en nombre d'instances résolues, ni en temps moyen de calcul. Notre critère DISTANCE gagne légèrement.

Un tableau d'instances non résolues par au moins un critère en moins d'une heure est également présenté dans le tableau 7.5.

Comparaison avec la fréquence prudente

Enfin, nous comparons les cinq critères avec FréqPrudente que nous avons défini, une fréquence géométrique qui garde les clauses apprises avec une durée plus longue.

	Minisat2.2-F3	
Stratégies	#Résolues (#SAT-#UNSAT)	tps moy
CSIDS	192 (91-101)	1611.71s
LBD	197 (**95**-102)	1556.56s
PSM	191 (92-99)	1571.07s
BTL	196 (94-102)	1577.00s
DISTANCE	**198** (93-**105**)	**1514.18s**

TABLE 7.6 – Comparaison avec les critères de état-de-l'art intégrés dans MiniSAT2.2 avec FréqPrudente

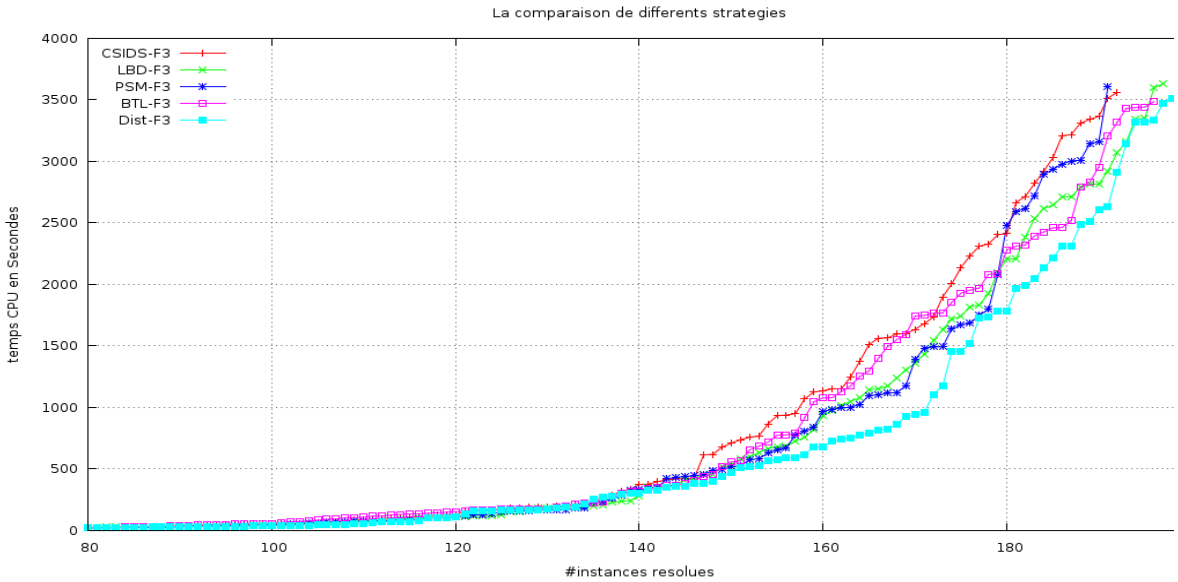


FIGURE 7.5 – Comparaison avec les critères de état-de-l'art intégrés dans MiniSAT2.2 avec FréqPrudente

Instance	Statues	CSIDS	LBD	PSM	BTL	DISTANCE
1dlx-c-iq60-a	UNSAT	3206.73	2813.92	-	3436.87	**2634.37**
aaai10-planning-ipc5-pathways-17-step21	SAT	-	**756.03**	-	-	-
aes-32-3-keyfind-1	SAT	-	-	-	-	**591.76**
blocks-blocks-36-0.120-NOTKNOWN	UNSAT	2403.13	**1715.15**	3158.90	-	1784.68
comb1.shuffled	UNSAT	-	-	-	**2954.96**	-
countbitswegner128	UNSAT	**2136.86**	-	-	-	3508.15
dated-10-17-u	UNSAT	-	**2618.69**	3003.15	2789.20	3141.32
E00N23	UNSAT	**3310.27**	-	-	-	-
gss-22-s100	SAT	-	3159.91	2718.74	683.57	**17.78**
minandmaxor128	UNSAT	3514.65	-	-	-	**3338.13**
partial-10-13-s	SAT	2818.97	-	1115.42	3431.45	**675.89**
partial-10-17-s	SAT	**419.12**	1172.38	-	-	-
rand-net70-60-10.shuffled	UNSAT	-	3069.46	-	**3316.71**	-
SAT-dat.k100	UNSAT	**2003.30**	-	-	-	2313.22
SAT-dat.k45	UNSAT	**2232.89**	3344.81	2615.65	-	2607.74
SAT-dat.k80	UNSAT	-	2709.82	-	3442.28	**2130.63**
slp-synthesis-aes-bottom13	UNSAT	-	2787.32	1492.55	**1250.91**	1520.85
slp-synthesis-aes-bottom14	UNSAT	-	-	**1671.14**	2279.33	-
slp-synthesis-aes-top25	SAT	-	3349.29	-	**3203.53**	-
slp-synthesis-aes-top26	SAT	3560.77	**1142.51**	1637.56	2310.26	-
smtlib-qfbv-aigs-countbits128-tseitin	UNSAT	2919.24	-	-	**2828.20**	-
smtlib-qfbv-aigs-servers-slapd-a-vc149789-tseitin	SAT	-	**2384.60**	-	2393.91	-
smtlib-qfbv-aigs-VS3-benchmark-S2-tseitin	UNSAT	-	2814.92	-	-	**2509.34**
sokoban-sequential-p145-microban-sequential.050-NOTKNOWN	SAT	-	-	-	**3484.32**	-
sokoban-sequential-p145-microban-sequential.070-NOTKNOWN	SAT	-	-	**982.25**	-	-

TABLE 7.7 – 2011 SAT Compétition, Industriel : temps (s) résultats 5 critères avec la fréquence prudente

Instance	**Statues**	**CSIDS**	**LBD**	**PSM**	**BTL**	**DISTANCE**
sortnet-8-ipc5-h19-sat	SAT	-	-	**486.70**	2459.10	1452.44
transport-transport-city-sequential-35nodes-1000size-4degree-100mindistance-4trucks-14packages-2008seed.030-NOTKNOWN	SAT	-	**1306.71**	3009.81	-	-
transport-transport-three-cities-sequential-14nodes-1000size-4degree-100mindistance-4trucks-14packages-2008seed.020-NOTKNOWN	SAT	3366.22	-	-	-	**2488.22**
UR-20-10p0	UNSAT	-	3629.93	-	-	**3319.50**
UR-20-10p1	SAT	-	-	-	-	**3470.02**
vmpc-32.renamed-as.sat05-1919	SAT	195.77	532.27	-	-	**15.12**
vmpc-34.renamed-as.sat05-1926	SAT	-	-	-	**1492.39**	-
vmpc-35.renamed-as.sat05-1921	SAT	-	**2532.44**	-	-	-
x1mul.miter.shuffled-as.sat03-359	UNSAT	-	-	3608.71	2517.59	**1729.90**

TABLE 7.8 – 2011 SAT Compétition, Industriel : temps (s) résultats 5 critères avec la fréquence prudente

Cette fois, notre critère DISTANCE est encore le meilleur, il résout 198 instances au total (93 SAT et 105 UNSAT) (voir le tableau 7.6). Il est aussi le plus rapide (voir la Figure 7.5). Cependant, le critère LBD a aussi une bonne performance, il résout seulement une instance de moins et plus d'instances SAT(95).

De manière similaire, une liste (7.7 et 7.8) des instances non résolue par au moins un critère en une heure est donné.

7.5.2 Intégration dans MANYSAT

L'une des principales forces des solveurs parallèles de type portfolio réside dans le choix des stratégies diversifiantes. À partir des résultats obtenus jusqu'à présent, se baser sur les stratégies de nettoyage pour construire un portfolio est une perspective intéressante au vue de la complémentarité remarqué à travers les résultats expérimentaux.

En analysant les performances des stratégies que nous avons présentées précédemment, nous avons choisi les 4 stratégies suivantes : CSIDS, LBD, BTL et DISTANCE avec la fréquence FreqPrudente qui nous apparaît la plus approprié. Nous allons maintenant intégrer ces quatre stratégies dans MANYSAT (chaque cœur utilise une stratégie différente).

Stratégies	#Résolues (#SAT-#UNSAT)	tps moy
Manysat2.0	213 (99-114)	1339.63s
Manysat2.0+4bases	**219 (103-116)**	**1267.07s**

TABLE 7.9 – Comparaison de Manysat2.0 et Manysat2.0 avec des stratégies de réduction différentes

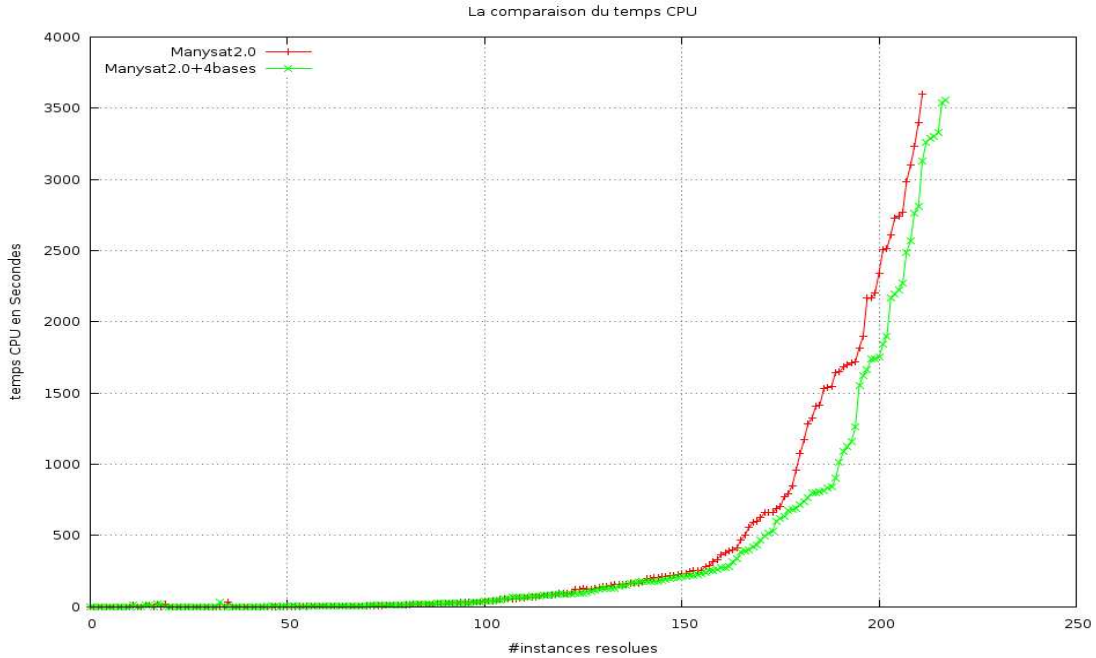


FIGURE 7.6 – Comparaison de Manysat2.0 et Manysat2.0 avec des stratégies de réduction différentes

La figure 7.6 et le tableau 7.9 montrent que diversifier les stratégies de nettoyage de la base des clauses apprises, est un bon moyen permettant d'obtenir des résultats nettement améliorées.

7.6 Conclusion

Dans ce chapitre, nous avons étudié une composante très importante des solveurs SAT modernes à savoir la stratégie de réduction de la base des clauses apprises. Motivé par les résultats compétitifs d'une comparaison entre un critère simple (aléatoire) et ceux utilisés par les solveurs de l'état de l'art, nous avons introduit deux nouveaux critères permettant d'identifier les clauses pertinentes pour la suite de la recherche.

Ces deux critères (plutôt statiques même s'ils peuvent être ajustés dynamiquement) visent à garder les clauses qui permettent d'élaguer l'espace de recherche. Nos critères sont basés sur les niveaux d'affectations des littéraux de la clause apprise. Cela permet d'identifier les clauses apprises qui pourront être utilisées dans le processus de propagation en haut de l'arbre de recherche.

Afin de justifier la pertinence de nos critères, nous avons procédé à une évaluation des stratégies de réduction avec les différentes fréquences. Nous avons démontré expérimentalement que nos deux nouveaux critères sont meilleurs que ceux de l'état de l'art.

Nous avons montré aussi qu'en diversifiant les stratégies de nettoyages de la base de clauses apprises dans un solveur parallèle de type portfolio, la performance du solveur est bien meilleure.

En perspective, nous envisageons de mener une étude pour déterminer le nombre de clauses apprises à éliminer à chaque étape. Dans le cadre parallèle, les clauses apprises sont échangées en fonction de la taille. Trouver d'autres critères pour déterminer les clauses les plus pertinentes à échanger est une perspective prometteuse.

Chapitre 8

Approche contrôle virtuel basée sur les Chaînes d'équivalence

Sommaire

DANS CE CHAPITRE, nous proposons une stratégie qui consiste à encadrer les threads dans un solveurs parallèle en divisant virtuellement l'espace de recherche avec les chaînes d'équivalences. Par rapport à la vraie division avec les chemins de guidage, la division virtuel grâce aux chaînes d'équivalences est moins contraignante : le contrôle virtuel. Nous ne forçons pas le solveur à suivre la direction des contraintes ajoutées. Au lieu d'attacher des contraintes au solveur, nous souhaitons seulement satisfaire des contraintes distribuées aux threads. Cela nous permet de trouver un compromis entre la contrôle forcé par le chemin de guidage et un contrôle moins contraignant pour le solveur séquentiel. C'est un contrôle plus léger par rapport à l'approche diviser pour régner.

L'idée de notre nouvelle approche est de combiner les avantages de l'approche *diviser pour régner* et ceux de l'approche *portfolio* Hamadi *et al.* (2009d), Guo *et al.* (2010).

Ce chapitre est organisé de la manière suivante. Nous introduisons d'abord la motivation ayant conduit à la naissance de notre stratégie, ensuite nous présentons notre stratégie appelée *contrôle virtuel*, qui encadre les threads en utilisant les chaînes d'équivalences entre littéraux. Avant de conclure et donner quelques perspectives, des résultats expérimentaux préliminaires sont donnés afin d'évaluer le comportement de notre nouvelle approche.

8.1 Motivation

Il existe principalement deux types de solveur SAT parallèles. Le premier type de solveurs est basé le principe de *diviser pour régner*. Il s'agit de soit diviser l'espace de recherche en utilisant par exemple le *chemin de guidage*, soit décomposer la formule originale en sous formules de tailles plus réduites.

La deuxième approche est de type *portfolio*. Ce type de solveur utilise un portfolio complémentaires d'algorithmes séquentiels conçus grâce à une minutieuse variation dans les principales stratégies de résolution. Ces stratégies sont combinés au partage des clauses apprises dans le but d'améliorer la performance globale. Ils évitent le problème de distribution de la charge entre les processeurs en laissant plusieurs moteurs DPLL travailler ensemble en mode compétition/coopération pour être le premier à

résoudre le problème. Chaque moteur (solveur séquentiel) travaille sur la formule originale, l'espace de recherche n'est plus divisé ni décomposé.

Les deux types de solveurs ont leur avantages et les désavantages.

La difficulté intrinsèque au solveur de type *diviser pour régner* est l'équilibrage de charge entre les unités de calculs. En plus, les sous arbres de recherche divisés à l'aide du chemin de guidage conduisent à l'exploration des espaces de recherche non liées. Celui-ci va réduire l'impact de l'échange des clauses apprises.

D'un autre coté, l'inconvénient du solveur de type *portfolio* est sa scalabilité. Une fois le nombre de threads (la taille du portfolio) devient grande, il sera très difficile de trouver des stratégies bien différenciées et performantes pour tous les moteurs.

8.2 Le contrôle virtuel avec chaînes d'équivalences

Les méthodes de diviser pour régner consistent à imposer un chemin de guidage aux threads, cela force chaque thread à suivre son chemin reçu. Un grand inconvénient de ce type de méthodes est l'échange des clauses apprises entre les threads, puisque pour chaque thread, il faut bien distinguer les clauses apprises réduites seulement par la formule originale et celles par la formule originale avec les clauses dans le chemin de guidage.

Notre méthode, le contrôle virtuel, consiste à ajouter les contraintes au solveur mais le solveur ne les utilise pas dans le processus de propagation unitaire.

Algorithme 8.1 : Le contrôle virtuel

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

```
1  F2 ⟵ CréerContraintes();   /* l'ensemble de contraintes ajouté */
2  VirtuelContrôle ⟵ true ;
3  tant que (true) faire
4      PropagationUnitaire();
5      si (pas de conflit) alors
6          si (VirtuelContrôle ==true) alors VirtuelPropager();
7          si (il y a virtuel conflit) alors
8              MettreAJourVirtuelContrôle();
9              si (VirtuelContrôle ==false) alors break ;
10             AnalyseVirtuelConflit();
11             au lieu de propager un littéral on prend une nouvelle décision marque la décision ;
12             RetourArrière();
13         PrendreUneDécision;
14         si (toutes les variables sont affectées) alors retourner SAT ;
15     sinon
16         MettreAJourVirtuelContrôle();
17         si dl = 0 alors retourner UNSAT ;
18         AnalyseConflit();
19         (ajoute la clause apprise à la base) ;
20         si (la decision est marqué) alors AnalyseVirtuelConflit();
21         RetourArrière();
```

L'algorithme 8.1 désigne notre méthode contrôle virtuel intégré dans un solveur CDCL. Nous commençons d'abord par créer les contraintes à ajouter au solveur (ligne 1). Pour faire cela, nous avons choisi d'ajouter des chaînes d'équivalences.

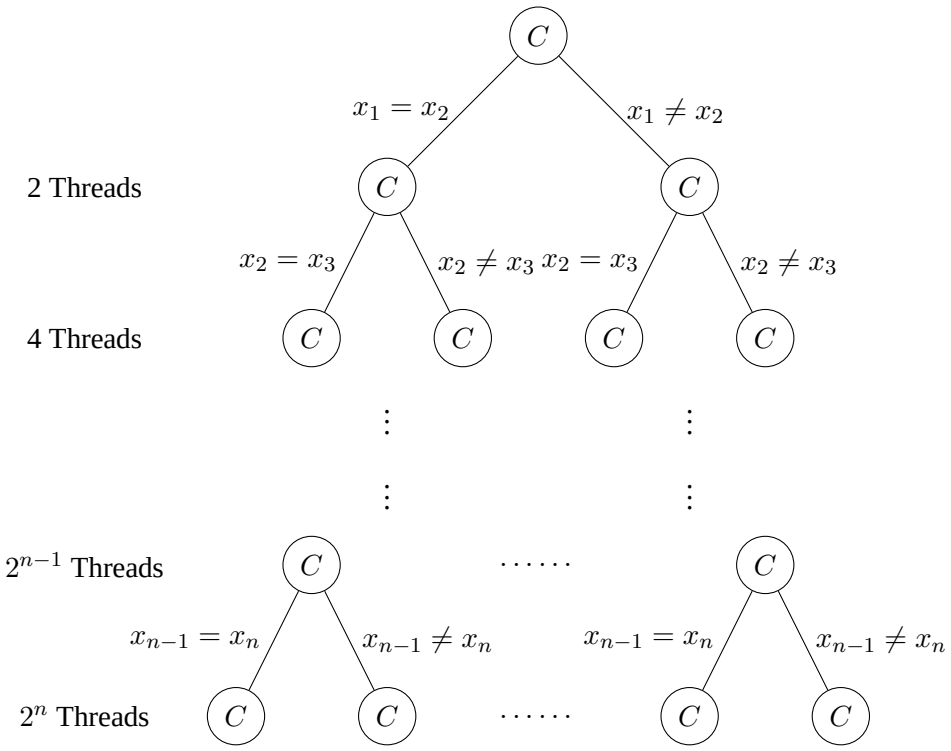


FIGURE 8.1 – L'attribution des chaînes d'équivalences

S'il y a quatre unités de calcul, nous allons choisir 3 variables x_1, x_2, x_3, et les contraintes ajouté au 4 threads seront respectivement :

1. $x_1 = x_2, x_2 = x_3$,
2. $x_1 = x_2, x_2 \neq x_3$,
3. $x_1 \neq x_2, x_2 = x_3$
4. $x_1 \neq x_2, x_2 \neq x_3$.

Dans le cas de 8 threads, le nombre de variables à contrôler devient 4. Les contraintes ajoutés à chaque thread seront respectivement :

1. $x_1 = x_2, x_2 = x_3, x_3 = x_4$
2. $x_1 = x_2, x_2 = x_3, x_3 \neq x_4$,
3. $x_1 = x_2, x_2 \neq x_3, x_3 = x_4$,
4. $x_1 = x_2, x_2 \neq x_3, x_3 \neq x_4$,
5. $x_1 \neq x_2, x_2 = x_3, x_3 = x_4$
6. $x_1 \neq x_2, x_2 = x_3, x_3 \neq x_4$,
7. $x_1 \neq x_2, x_2 \neq x_3, x_3 = x_4$,

8. $x_1 \neq x_2, x_2 \neq x_3, x_3 \neq x_4$,

La figure 8.1 illustre dans le cas général, comment nous distribuons les chaînes d'équivalences aux threads. Notons qu'avec l'augmentation du nombre de threads, les chaînes d'équivalences deviennent plus longue. Ce qui peut poser un problème de synchronisation. Pour éviter ce genre de problèmes, nous proposons ici une autre méthode de distribution des contraintes.

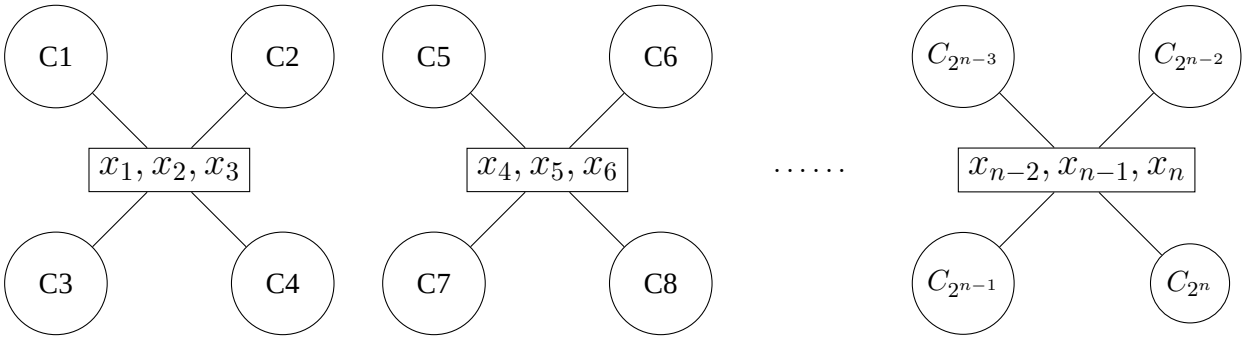


FIGURE 8.2 – Schéma de groupe de 4 cœurs

Comme indiqué dans la figure 8.2, nous divisons les threads en petit groups de 4 threads. Pour chaque groupe, 3 variables sont sélectionnées pour le contrôle virtuel, et la synchronisation seront fait entre les 4 threads.

Une fois les contraintes sont crées, nous les ajoutons aux threads et avec un booléen *VirtuelContrôle* pour démarrer et arrêter le contrôle (ligne 2). Il faut noter que dans notre méthode, les contraintes ajoutées ne participent pas au processus de propagation unitaire. Notre objectif est de définir un cadre de recherche pour chaque solveur séquentiel dans le portfolio. Afin d'assurer que chaque moteur ne sort pas du cadre que nous avons défini, avant que le solveur ne prenne une décision, nous vérifions la satisfaction de toutes les contraintes ajoutées (ligne 6). S'il y a des contraintes falsifiées (ligne 7-12), nous allons analyser ce conflit virtuel, à partir du processus *AnalyseConflitVirtuel*. Comme c'est un conflit virtuel provoqué par les contraintes ajoutés, nous ne pouvons pas propager le littéral assertif, mais nous choisissons ce littéral comme un littéral de décision. A cette décision, nommée *décision propagée*, est attribuée une raison et sera marquée comme propagée virtuellement.

Notons que le processus "AnalyseConflitVirtuel" ajoute dans le graphe d'implications un autre type de littéraux. Il y aura 3 types de littéraux : les littéraux propagées avec raisons , les décisions sans raisons et les décisions propagées avec raisons.

Quand un vrai conflit est rencontré (ligne 16-21), nous effectuons d'abord le processus d'analyse de conflits afin d'apprendre une clause apprise classique et l'ajouter à la base des clauses apprises. Ensuite dans le cas où la décision du niveau de conflit est marquée, nous effectuons également l'analyse du conflit virtuel. Cela nous permet d'améliorer parfois le niveau de retour arrière.

Nous allons expliquer en détail dans la section suivante les nouvelles fonctions de notre algorithme de contrôle virtuel.

8.3 Les fonctions importantes dans la stratégie de contrôle virtuel

Dans ce chapitre, nous allons détailler notre méthode de contrôle virtuel qui combine les avantage des approches diviser pour régner et portfolio. Notre approche permet d'encadrer la recherche comme les

approches diviser pour régner et échanger des clauses apprises librement comme dans un cadre portfolio.

8.3.1 Encadrer la recherche : Créer des Contraintes pour chaque thread

Ce processus consiste à :

1. sélectionner les variables à contrôler ,
2. créer des contraintes complémentaires à partir des variables sélectionnés, attribuer les clauses aux threads

Pour chaque thread, on ajoute une variable booléenne afin de synchroniser le contrôle entre les threads. Dans un premier temps, les variables sont sélectionnées dans un certain ordre. Comme indiqué précédemment, les contraintes distribuées aux threads sont des chaînes d'équivalences de la forme $a = b$. Deux clauses binaires $\bar{a} \vee b$ et $\bar{b} \vee a$ seront ajoutés pour chaque équivalence $a = b$ afin d'encadrer la recherche de chaque thread. Comme les clauses attribuées aux threads ne participent par au processus de propagation unitaire, nous pouvons échanger les clauses apprises comme dans un cadre portfolio.

8.3.2 Synchronisation : Mettre à jour le contrôle virtuel

Cette fonction consiste à reproduire les contraintes et synchroniser le contrôle pour les threads.

Une fois les contraintes ajoutées sont réfutées par un thread, son booléen VirtuelContrôle est mis à faux et le solveur procède à l'arrêt du contrôle et continue la recherche. Dès qu'il reste qu'un seul thread, il fait le contrôle ou le contrôle est effectué pendant un certain nombre de conflits. Le thread 1 (nous l'avons sélectionné pour effectuer la synchronisation) va créer de nouvelles contraintes et démarrer un nouveau contrôle de son groupe. Cette fonction est pour l'instant appelée à chaque redémarrage.

8.3.3 Analyse de Conflits Virtuels

Le processus AnalyseConflitVirtuel est le processus d'analyse de conflit avec certaines modifications.

Comme indiqué précédemment, pour chaque thread, les contraintes ajoutées ne sont pas utilisées par le processus de propagation unitaire. Nous vérifions à chaque fois avant de prendre une décision, la satisfaction des contraintes ajoutées. Une fois qu'un conflit virtuel est rencontré, nous allons analyser ce conflit virtuel, apprendre une clause apprise et un littéral assertive. Au lieu de propager le littéral assertive, nous allons prendre une décision sur ce littéral et lui associer une raison, ce genre de décisions nous les appelons *décision propagée*

Si la décision courante est une décision propagée, l'analyse se poursuit jusqu'au niveau dont la décision est une décision classique, mais pas une décision propagée. Ceci vient du fait que les contraintes distribuées aux threads ne participent pas à la propagation mais on veut les utiliser pour encadrer les threads.

Une fois qu'on rencontre un vrai conflit ou un conflit virtuel, une vérification des décisions sera effectuée pour déterminer si toutes les décisions (de niveau 0 jusqu'au niveau courant) sont des décisions propagées. Le booléen VirtuelContrôle sera mis à faux et le contrôle virtuel sera arrêté.

8.4 Expérimentations

Afin d'étudier l'impact de notre approche, nous avons implémenté notre approche dans le solveur Manysat2.0 et lancé les expérimentations sur 4 , 8 et 32 threads.

L'ensemble d'expérimentations (sur 4 et 8 threads) ont été menées sur un bi-processeurs Quad-Core Intel XEON X5550 avec 32Gb de mémoire. Les résultats sur 32 threads ont été lancé sur une machine de 8 processeurs, chaque processeur est un Quad-Core, la machine possède 256Gb de mémoire. 300 instances ont été utilisées, elles sont issues de la compétition SAT2011, catégorie industrielle. Pour l'ensemble des expérimentations le temps est limité à 30 minutes soit 1800 secondes.

	Manysat2.0	Manysat2.0+VC
Nb. threads	#Résolues (#SAT-#UNSAT)	#Résolues (#SAT-#UNSAT)
4 threads	196 (97-99)	193 (97-96)
8 threads	208 (100-108)	208 (103-105)
32 threads	224 (107-117)	223 (108-115)

TABLE 8.1 – Comparaison de scalabilité entre Manysat2.0 et Manysat2.0 intégrant notre approche de contrôle virtuel

Le tableau 8.1 montre le résultats de Manysat2.0 et notre approche de contrôle virtuel. Notons qu'en augmentant le nombre de threads, notre méthode semble compétitive.

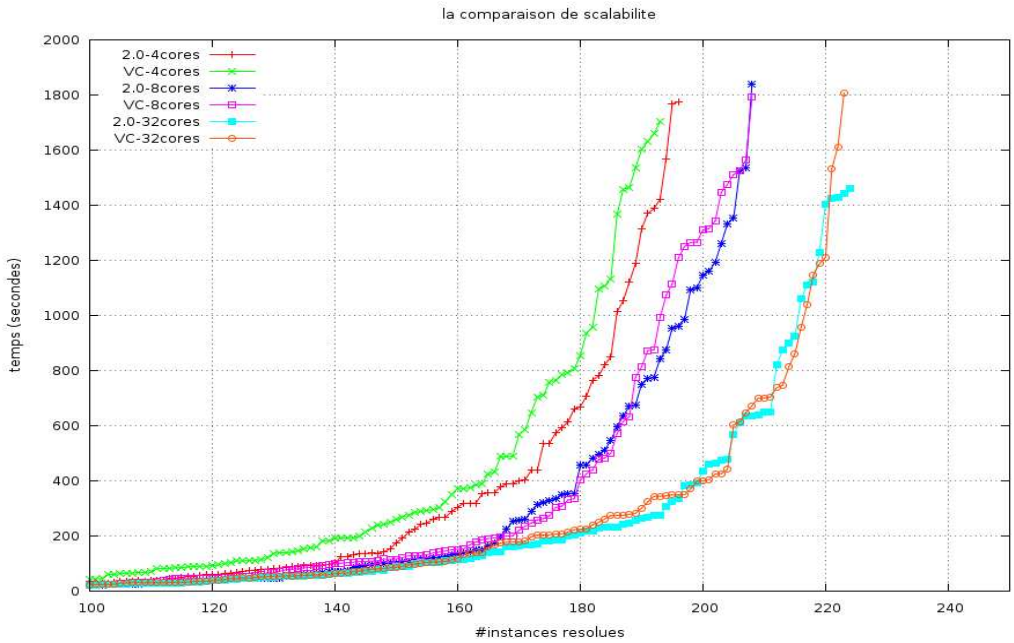


FIGURE 8.3 – Nombre d'instances résolues en fonction du temps

La figure 8.3 illustre l'évolution intéressante des performances entre Manysat2.0 et Manysat2.0 intégrant notre méthode de contrôle virtuel. Au départ, avec 4 threads, le contrôle virtuel est loin derrière

Manysat2.0, il s'approche quand dans le cas de 8 threads, et finalement avec 32 threads, le virtuel contrôle devient meilleur que Manysat2.0.

8.5 Conclusion

Dans ce chapitre , nous avons présenté une nouvelle approche de partitionnement de l'espace de recherche en utilisant les chaînes d'équivalence dans un solveur parallèle. Notre approche diffère des autres méthodes de type diviser pour régner. En effet, nos contraintes ne sont pas ajoutées vraiment au solveur (le processus de propagation n'utilise pas les contraintes ajoutées), et les solveurs ne sont pas dirigés mais contrôlés par les contraintes distribuées. Cette méthode nous permet non seulement de diviser l'espace de recherche mais aussi de laisser chaque thread à faire sa recherche librement et échanger leurs clauses apprises comme dans le cadre portfolio.

Plusieurs perspectives intéressantes s'ouvrent pour les travaux futurs. Premièrement, une fois les contraintes sont réfutées par un thread, c'est vraiment dommage de ne pas ajouter vraiment la négation des contraintes réfutées à la formule. Cette question constitue une perspective à court terme. De plus, le choix des variables pour créer les contraintes à ajouter aux solveurs et la fréquence de changement des contraintes pendant la recherche sont très importantes et vont directement influencer la qualité de la division. Étudier plus finement ces deux points constitue une perspective prometteuse.

Conclusion et Perspectives

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

Après avoir introduit dans la première partie, les différents principes nécessaire à la compréhension de cette thèse, nous avons proposé dans la deuxième partie, différentes améliorations dans le cadre de la résolution parallèle du problème SAT.

Nous avons d'abord exploré les principes de diversification et d'intensification dans le cadre de la résolution parallèle du problème SAT de type portfolio. Les deux concepts jouent un rôle important dans certain algorithmes de recherche comme la recherche locale, et semble être le point clé du solveurs SAT parallèle. Pour étudier ce compromis de diversification et d'intensification, nous avons défini deux rôles pour les unités de calcul, certaines d'entre elles, désignées comme maître, ont leur propres stratégies de recherche, et ont pour but d'assurer la diversification. D'autres classifiés comme « Esclaves » ont pour but d'intensifier les stratégies de leurs maîtres. Nous avons proposé différents types d'informations qu'un maître peut utiliser pour diriger un eslave. Notre résultat est que forcer l'esclave à brancher sur les variables du graphe d'implications dans un ordre donnée, permet d'améliorer significativement le solveur parallèle ManySAT.

Nous avons ensuite proposé une méthode permettant d'ajuster dynamiquement l'heuristique de choix de polarité des moteurs séquentiels dans le cadre d'un solveur de type portfolio. Nous nous sommes appuyés sur les heuristiques de choix de polarité de chaque moteur séquentiel afin de déterminer si deux d'entres eux effectuent une tâche similaire. Lorsqu'une telle situation se produit, nous proposons simplement d'inverser l'heuristique de choix de polarité de l'un des deux moteurs séquentiels. Nous espérons ainsi diversifier au mieux l'espace de recherche parcouru par les différents cœurs.

Notre troisième contribution concerne la gestion de la base de clauses apprises dans les solveurs SAT séquentiels. Nous avons proposé deux nouveaux critères pour identifier les clauses apprises pertinentes. Nous avons aussi étudié deux autres questions importantes dans une stratégie de réduction : la fréquence de réduction et le nombre de clauses supprimées à chaque réduction. L'expérimentation nous a montré que les stratégies peuvent s'avérer complémentaires. Nous avons alors proposé d'utiliser différentes stratégies de réduction de la base de clauses apprises pour les moteurs séquentiels dans un solveur parallèle du type portfolio.

La dernière contribution de cette thèse a porté sur la proposition d'une nouvelle approche de partitionnement de l'espace de recherche en utilisant les chaînes d'équivalences dans un solveur parallèle. Contrairement aux autres méthodes basées sur l'ajout de contraintes, dans notre approche, les contraintes ne sont pas ajoutées à la formule. En effet, afin d'éviter d'engendrer par analyse de conflits beaucoup de clauses dépendantes des contraintes ajoutées, dans notre approches ces contraintes sont uniquement utilisées pour contrôler la propagation.

Ces différentes contributions ouvrent de nombreuses perspectives. Premièrement, dans le cadre d'approches de type portfolio, au sujet de la diversification et de l'intensification, nous souhaitons adapter dynamiquement le nombre de maîtres/esclaves ainsi que les topologies selon la difficulté d'une instance donnée. Il semble que, pour les instances plus difficiles, associer plusieurs esclaves à un même maître permetterait d'obtenir de meilleurs performances sur les instances insatisfiables. En revanche, diminuer le nombre d'esclaves permet d'être plus efficace sur les instances satisfiables. En ce qui concerne l'ajustement dynamique de l'heuristique de choix de polarité, il semble que la fréquence selon laquelle l'ajustement est effectué influe sur les performances. Il est alors nécessaire d'étudier plus finement ce phénomène et de définir une stratégie dynamique pour régler ce paramètre.

Concernant la gestion de la base de clauses apprises, trouver de nouveaux critères pour déterminier le nombre de clauses apprises à éliminer à chaque réduction semble très important et intéressant à explorer. Il est aussi intéressant d'utiliser des critères différents pour contrôler plus finement l'échange des clauses apprises entre les différentes unités de calcul.

Finalement, la dernière contribution portant sur le partitionnement de l'espace de recherche de chaque moteur séquentiel dans un solveur parallèle en utilisant les chaînes d'équivalence, il serait intéressant de poursuivre le travail sur le choix des variables pour créer les contraintes.

L'échange d'informations est une question incontournable dans les solveurs parallèles, une fois le nombre des unités de calcul dans un solveur devient grand, l'échanges des clauses entre les threads sera très coûteux (une clique de n threads avec $n > 100$, on peut imaginer !). Alors, concevoir une architecture puissante et efficace pour l'échange des clauses pressante. En plus, la façon d'échange des clauses entre les threads comme nous avons indiqué dans Chapitre 7 est aussi une piste très intéressant à découvrir.

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.

Josep ARGELICH, Daniel LE BERRE, Inês LYNCE, João P. MARQUES SILVA et Pascal RAPICAULT : Solving linux upgradeability problems using boolean optimization. *In* Inês LYNCE et Ralf TREINEN, éditeurs : *Proceedings First International Workshop on Logics for Component Configuration*, volume 29, pages 11–22, 2010.

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, Benoît HOESSEN, Saïd JABBOUR, Jean-Marie LAGNIEZ et Cédric PIETTE : Revisiting clause exchange in parallel sat solving. *In SAT*, pages 200–213, 2012.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : On freezeing 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.

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

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

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 : Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

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 : Lazy hyper binary resolution. Technical report, 2009a.

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

Armin BIERE : Lingeling, plingeling, picosat and precosat at sat race 2010. Rapport technique, August 2010.

Armin BIERE, Alessandro CIMATTI, Edmund M. CLARKE, Masahiro FUJITA et Yunshan ZHU : Symbolic model checking using sat procedures instead of bdds. *In Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999a. ACM.

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, 1999b.

Armin BIERE, Marijn J. H. HEULE, Hans van MAAREN et Toby WALSH, éditeurs. *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications*. IOS Press, février 2009. ISBN 978-1-58603-929-5.

Wolfgang BLOCHINGER, Carsten SINZ et Wolfgang KÜCHLIN : Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Comput.*, 29:969–994, July 2003. ISSN 0167-8191.

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.

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.

Wahid CHRABAKH et Rich WOLSKI : Gridsat : A chaff-based distributed sat solver for the grid. *In Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 37–, New York, NY, USA, 2003. ACM. ISBN 1-58113-695-1.

Geoffrey CHU et Peter J. STUCKEY : Pminisat : A parallelization of minisat 2.0. Solver description, sat-race 2008, 2008.

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.

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.

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.

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.

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

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 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 et Jean-Marie LAGNIEZ : Ajustement dynamique de l'heuristique de polarité dans le cadre d'un solveur sat parallèle. *In Septièmes Journées Francophones de Programmation par Contraintes (JFPC11)*, JFPC11, pages 155–161, juin 2011a.

Long GUO et Jean-Marie LAGNIEZ : Dynamic polarity adjustment in a parallel sat solver (to appear). *In 23rd IEEE International Conference on Tools with Artificial Intelligence*, ICTAI'11, Nov. 7-9, 2011, Boca Raton, Florida, USA, novembre 2011b.

Youssef HAMADI, Said JABBOUR, Cédric PIETTE et Lakhdar SAÏS : Concilier parallélisme et déterminisme dans la résolution de sat. *In Journées Francophones de la Programmation par Contraintes(JFPC'11)*, page A paraitre, Lyon, jun 2011.

Youssef HAMADI, Saïd JABBOUR et Lakhdar SAIS : Control-based clause sharing in parallel sat solving. *In IJCAI*, pages 499–504, 2009a.

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, 2009b. IEEE Computer Society.

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

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

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.

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

Bernard JURKOWIAK, Chu Min LI et Gil UTARD : Parallelizing satz using dynamic workload balancing. *Electronic Notes in Discrete Mathematics*, 9:174–189, 2001.

Bernard JURKOWIAK, Chu Min LI et Gil UTARD : A parallelization scheme based on work stealing for a class of sat solvers. *J. Autom. Reasoning*, 34(1):73–101, 2005.

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.

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.

Stephan KOTTLER : Sartagnan : solver description. Solver description, sat-race 2010, 2010a.

Stephan KOTTLER : Sat solving with reference points. *In SAT*, pages 143–157, 2010b.

Jean-Marie LAGNIEZ : *Satisfiabilité proprositionnelle et raisonnement par contraintes : modèles et algorithmes*. Thèse doctorat, Université d'Artois, Centre de Recherche en Informatique de Lens, CRIL CNRS UMR 8188, décembre 2011.

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.

D. LE BERRE et A. PARRAIN : Sat4j : Bringing the power of sat technology to the java platform. http ://www.sat4j.org.

Daniel LE BERRE : Sat4j, un moteur libre de raisonnement en logique propositionnelle, dec 2010.

Daniel LE BERRE et Pascal RAPICAULT : Dependency management for the eclipse ecosystem : eclipse p2, metadata and resolution. *In Proceedings of the 1st international workshop on Open component ecosystems*, pages 21–30, New York, NY, USA, 2009. ACM.

Daniel LE BERRE et Olivier ROUSSEL : « 12th international symposium on theory and applications of satisfiability testing ». May 2009.

Matthew D. T. LEWIS, Tobias SCHUBERT et Bernd BECKER : Early conflict detection based sat solving. *In MBMV*, pages 243–249, 2004.

Matthew D. T. LEWIS, Tobias SCHUBERT et Bernd BECKER : Multithreaded sat solving. *In ASP-DAC*, pages 926–931, 2007.

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.

Gordon E. MOORE : Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86 (1):82–85, 1998.

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.

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 : 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, 2009. Springer-Verlag. ISBN 978-3-642-02776-5.

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.

William V. QUINE : *Methods of logics*. Henry Holt, 1950.

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.

Olivier ROUSSEL : Description of ppfolio. Rapport technique, 2011. SAT 2011 Competition Event Booklet, http://www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf.

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.

Carsten SINZ, Wolfgang BLOCHINGER et Wolfgang KÜCHLIN : Pasat – parallel sat-checking with lemma exchange : Implementation and applications. *Electronic Notes in Discrete Mathematics*, 9:205 – 216, 2001. ISSN 1571-0653. LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001).

Mate SOOS : Cryptominisat. Rapport technique, available on WebSite : http://www.msoos.org, 2011.

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.

M.Lewis T.SCHUBERT et B.BECKER : Antom : solver description. Solver description, sat-race 2010, 2010.

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.

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.

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 2003.

Lin XU, Frank HUTTER, Holger H. HOOS et Kevin LEYTON-BROWN : Satzilla : portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008. ISSN 1076-9757.

Y.HAMADI et C.M.WINTER : Seven challenges in parallel sat solving. AAAI'12. AAAI Press, 2012. Invited paper, to appear.

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.

Hantao ZHANG, Maria Paola BONACINA et Jieh HSIANG : Psato : a distributed propositional prover and its application to quasigroup problems. *J. Symb. Comput.*, 21:543–560, June 1996.

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 Automated Deduction*, pages 295–313, London, UK, 2002. Springer-Verlag. ISBN 3-540-43931-5.

Résumé

Cette thèse porte sur la résolution séquentielle et parallèle du problème de la satisfiabilité propositionnelle (SAT). Ce problème important sur le plan théorique admet de nombreuses applications qui vont de la vérification formelle de matériels et de logiciels à la cryptographie en passant par la planification et la bioinformatique.

Plusieurs contributions sont apportées dans cette thèse. La première concerne l'étude et l'intégration des concepts d'intensification et de diversification dans les solveurs SAT parallèle de type portfolio.

Notre seconde contribution exploite l'état courant de la recherche partiellement décrit par les récentes polarités des littéraux « progress saving », pour ajuster et diriger dynamiquement les solveurs associés aux différentes unités de calcul.

Dans la troisième contribution, nous proposons des améliorations de la stratégie de réduction de la base des clauses apprises. Deux nouveaux critères, permettant d'identifier les clauses pertinentes pour la suite de la recherche, ont été proposés. Ces critères sont utilisés ensuite comme paramètre supplémentaire de diversification dans les solveurs de type portfolio.

Finalement, nous présentons une nouvelle approche de type diviser pour régner où la division s'effectue par ajout de contraintes particulières.

Mots-clés: SAT, CDCL, résolution parallèle, portfolio, diviser pour régner

Abstract

In this thesis, we deal with the sequential and parallel resolution of the problem SAT. Despite of its complexity, the resolution of SAT problem is an excellent and competitive approach for solving the combinatorial problems such as the formal verification of hardware and software, the cryptography, the planning and the bioinfomatics.

Several contribution are made in this thesis. The first contribution aims to find the compromise of diversification and intensification in the solver of type portfolio. In our second contribution, we propose to dynamically adjust the configuration of a core in a portfolio parallel sat solver when it is determined that another core performs similar work. In the third contribution, we improve the strategy of reduction of the base of learnt clauses, we construct a portfolio strategy of reduction in parallel solver. Finally, we present a new approach named "Virtual Control" which is to distribute the additional constraints to each core in a parallel solver and verify their consistency during search.

Keywords: SAT, CDCL, parallel solver, portfolio, divider and conquer