UFR des Sciences

Faculté Jean Perrin

De la Satisfaisabilité à la Compilation de Bases de Connaissances Propositionnelles

THÈSE

présentée et soutenue publiquement le 18 janvier 1999

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Bertrand Mazure

Composition du jury

Rapporteurs :	Jacqueline Chabrier	
	Pierre Siegel	
Examinateurs :	Gilles Goncalves	
	Éric Grégoire	(directeur de thèse)
	Pierre Marquis	
	Lakhdar Saïs	

Université d'Artois – Centre de Recherche en Informatique de Lens
rue de l'Université, SP 16 – F-62307 LENS Cedex
Tél : +33 (0)3 21 79 32 74 – Fax : +33 (0)3 21 79 32 72

Remerciements

Aucun travail ne s'accomplit dans la solitude. Aussi ai-je trouvé normal que figurent au début de cette thèse des remerciements adressés à ceux qui ont aidé et concouru à sa réalisation.

En premier lieu, je tiens à remercier les membres du jury,

Madame Jacqueline Chabrier, Maître de Conférences à l'Université de Bourgogne, Habilitée à diriger des recherches, Qualifiée par le CNU aux fonctions de professeur et Monsieur Pierre Siegel, Professeur à l'Université de Provence pour l'intérêt qu'ils ont porté à mes travaux en acceptant d'être les rapporteurs de cette thèse et pour les remarques et conseils éclairés qu'ils m'ont donnés et qui ont contribué à améliorer ce manuscrit.

Monsieur Gilles Goncalves, Professeur à l'Université d'Artois et Directeur du laboratoire organisation et génie de la production (LABOGP), pour s'être intéressé à mes travaux et avoir bien voulu participer à mon jury.

Monsieur Pierre Marquis, Professeur à l'Université d'Artois, pour ses précieux conseils et encouragements. Sans les longues et enrichissantes discussions que nous avons eues ensemble tout au long de la thèse, ce travail ne serait certainement pas devenu ce qu'il est.

Mes directeurs de thèse, Éric Grégoire qui m'enseigna la rigueur d'un travail de longue haleine et l'ABC d'une méthode de recherche et Lakhdar Saïs pour qui cette page ne suffirait pas à lui témoigner ma profonde gratitude. Tout au long de ces années, il a représenté beaucoup plus qu'un responsable de thèse et de véritables liens d'amitiés se sont noués au travers de cette collaboration étroite et quotidienne. Lakhdar je te dis simplement merci et « *plus vite que ça* ».

Je tiens également à remercier,

Les membres du CRIL, pour leur sympathie et pour leur convivialité. Je remercie particulièrement Sylvie Coste (la seule marseillaise à soutenir le RCL) avec qui j'ai partagé de nombreuses heures à administrer les machines du CRIL. Sans son dévouement, cette tâche ingrate n'aurait certainement pas pu être menée à bien et surtout pas aussi agréablement. Je lui exprime également ma profonde gratitude pour m'avoir soulagé de certaines tâches d'enseignement durant la rédaction de la thèse. Je tiens également à remercier Jean-Luc Coquidé pour sa gentillesse, son soutien et ses nombreux encouragements surtout dans les derniers moments, si difficiles mais si décisifs. Je remercie également Laure Brisoux en tant qu'utilisatrice critique mais constructive de mes programmes et pour la bonne ambiance qu'elle a su apporter dans notre bureau.

Ma famille et plus particulièrement mes parents et Valérie qui m'ont apporté réconforts et encouragements durant les moments de doute et m'ont prodigué tout au long de cette thèse affection et soutien enthousiaste. Pour cela et beaucoup d'autres choses je les remercie.

À ma grand-mère,

Table des matières

Table des figures

Table des algorithmes

Liste des tableaux

Liste des exemples

Introduction générale

L'un des axes majeurs de recherche en Intelligence Artificielle concerne la représentation des connaissances et le raisonnement à partir de celles-ci. Une approche assez naturelle repose sur l'utilisation des concepts de la logique mathématique. Cependant dans un cadre informatique, on attend des résultats pratiques : la déduction doit être programmée et produire des résultats dans un temps raisonnable.

Le choix d'un langage de représentation de la connaissance ainsi que la recherche d'outils et de méthodes de résolution qui allient « puissance d'expression et efficacité » sont des problèmes anciens (cf. par exemple [Brachman & Levesque 1982]) mais encore cruciaux à l'heure actuelle dans des domaines tels que la programmation logique avec contraintes et la démonstration automatique. La contrainte d'efficacité est souvent liée à une diminution de la puissance d'expression et inversement une « trop » grande expressivité se heurte au problème de l'explosion combinatoire lors de la résolution. Ainsi l'utilisation de logiques « non élémentaires » pose des problèmes tels que la dégradation extrêmement rapide des performances des méthodes de résolution dès lors que la représentation des connaissances prises en compte devient importante. Pour ces raisons, il nous a paru intéressant de circonscrire notre étude à la logique propositionnelle.

En dépit de son apparente simplicité, ce langage permet, lorsqu'il est utilisé dans sa totalité, de représenter de manière simple une grande variété de connaissances. De plus, d'autres logiques « plus puissantes » s'y ramènent naturellement ou le contiennent. En outre, de nombreux problèmes pratiques peuvent s'exprimer naturellement en logique propositionnelle, sans devoir passer par des logiques très riches qui parfois, masquent les difficultés théoriques et pratiques sous une inflation de concepts plus ou moins complexes.

L'efficacité des raisonnements, compte tenu du mode de représentation choisi, demeure bien évidemment le problème primordial. Il reste que le problème de la vérification de la cohérence (encore appelée abusivement, car ces termes sont des néologismes, consistance ou satisfaisabilité) d'une formule exprimée en logique propositionnelle n'admet pas de solution à la fois générale et efficace. En effet, ce problème, plus connu sous la forme du test de satisfaisabilité d'une formule propositionnelle mise sous forme normale conjonctive (**SAT**) est considéré comme l'archétype des problèmes NP-complets ; à ce titre il joue un rôle central en théorie de la complexité. Ainsi, aussi longtemps que la conjecture $P = NP$ n'a pas été démontrée « vraie », nous ne disposerons pas d'algorithmes toujours efficaces, c'est-à-dire de complexité au pire polynomiale pour résoudre ce problème.

Par ailleurs, la plupart des modèles de déduction des connaissances en logique requièrent au moins un test de satisfaisabilité. La majorité des systèmes en intelligence artificielle à base de contraintes réalisent explicitement ou implicitement ce test de satisfaisabilité. Ce dernier est donc crucial pour le développement pratique des systèmes à base de connaissances. Par exemple, le problème consistant à déterminer si une information f peut être déduite à partir de la connaissance exprimée par une formule $\mathcal{F}$ de la logique propositionnelle revient à tester l'insatisfaisabilité de la formule $(\mathcal{F} \wedge \neg f)$. Cependant dans diverses applications, il est courant que des déductions successives doivent être calculées pour une même base de connaissances. Dans ce contexte, effectuer un test de consistance pour chaque formule à déduire devient

très vite prohibitif au vu des considérations de complexité théorique et des performances pratiques des meilleurs algorithmes actuels de satisfaction. Ainsi le problème de la déduction ou de l'interrogation d'une base de connaissances s'inscrit dans un problème plus large de représentation des connaissances à partir desquelles la déduction doit être calculée de manière efficace. À cette fin, la compilation logique de bases de connaissances est l'une des stratégies les plus fréquemment utilisées.

L'objet de ce travail est l'étude des problèmes de satisfaisabilité et de compilations logiques de bases de connaissances exprimées en logique propositionnelle. Cette étude présente avant tout un objectif pratique : son ambition est de contribuer à l'amélioration des algorithmes existants et de proposer de nouvelles techniques afin d'élargir l'étendue des instances traitables.

La première partie de ce manuscrit situe le contexte de mes travaux. Elle concentre quelques rappels de logique propositionnelle, ainsi que de la théorie de la complexité. Ces rappels nous permettent de définir formellement le problème **SAT** et de mesurer pleinement sa difficulté. D'autre part, un large passage est dédié aux classes polynomiales de **SAT**. En effet, une partie de nos travaux est consacrée à l'exploitation de ces classes polynomiales dans le cadre de la compilation logique de bases de connaissances propositionnelles. La fin de cette partie introductive est dédiée à la présentation des problèmes gravitant autour du test de satisfaisabilité, comme les problèmes d'optimisation et de recherche associés à **SAT** ainsi que le problème de la déduction logique.

La seconde partie traite des aspects algorithmiques du problème **SAT**. Nous présentons dans cette partie les différentes études menées sur les algorithmes existant et proposons de nouvelles stratégies dans l'optique d'élargir les classes d'instances traitables en pratique. À l'heure actuelle deux grands principes de résolution surpassent en efficacité toutes les autres méthodes de leur catégorie respective : la recherche locale en ce qui concerne les approches incomplètes et la procédure de Davis & Putnam pour les méthodes complètes.

La recherche locale a contribué ces dernières années au regain d'intérêt constaté pour le problème **SAT**. Dans ce domaine nous proposons un nouvel algorithme de recherche locale basé sur une utilisation systématique d'une liste de réparations interdites (la méthode tabou). L'efficacité des méthodes de recherche locale est assujettie à un paramètrage pointilleux. L'algorithme proposé respecte tout naturellement cette règle. Cependant un résultat empirique surprenant permet de régler automatiquement la taille de la liste d'interdits en fonction du nombre de variables pour des instances *k***SAT** aléatoires.

La procédure de Davis & Putnam est quasiment au cœur de tous les solveurs complets pour **SAT**. Depuis maintenant près de quarante ans, cette procédure fait l'objet de multiples recherches visant à son amélioration. Ces améliorations portent essentiellement, soit sur l'heuristique de branchement, soit sur des techniques de simplification permettant d'élaguer l'arbre de recherche. Notre contribution concerne ce second point. Plus précisément, nous proposons une généralisation du théorème de partition du modèle et montrons comment une application restreinte à la propagation unitaire de cette généralisation permet de diminuer significativement le nombre de nœuds explorés par l'arbre de décision décrit par la procédure de Davis & Putnam.

La complémentarité des méthodes de recherche locale et de la procédure de Davis & Putnam amène naturellement à l'idée de faire coopérer ces méthodes afin de tirer avantage des qualités intrinsèques de chacune d'elles. Nous proposons plusieurs schémas de coopération entre ces méthodes et montrons que de telles combinaisons sont très performantes en pratique. Plus précisément, nous montrons comment la recherche locale peut être utilisée pour circonscrire l'inconsistance, alors qu'à l'origine ces méthodes sont dédiées à la recherche de solutions. L'un des schémas de coopération proposé utilise la recherche locale afin de prolonger l'interprétation partielle construite par la procédure de Davis & Putnam vers un modèle et comme heuristique de branchement pour cette procédure. Ce schéma rivalise et surpasse même les meilleurs algorithmes complets existants sur de nombreuses instances classiquement utilisées pour l'évaluation des algorithmes.

Nous terminons cette seconde partie par une étude réalisée sur le générateur standard d'instances aléatoires *k***SAT**. Nous proposons d'étendre ce générateur par l'ajout d'un nouveau paramètre : la probabilité de positiver un littéral. Empiriquement, nous avons mis en évidence un lien surprenant entre cette probabilité et le pic de difficulté des instances générées. Par ailleurs, nous avons étudié pour chacun des modèles de génération proposés le nombre de littéraux unitaires et de littéraux purs rencontrés au cours de la résolution de ces instances par une procédure classique de Davis & Putnam, ceci afin de déterminer les zones optimales d'utilisation des méthodes de simplification liées à ces littéraux particuliers.

La troisième partie de cette thèse est consacrée au problème de la déduction logique en calcul propositionnel et plus précisément à la compilation logique de bases de connaissances. Nous proposons une nouvelle technique de compilation logique préservant l'équivalence, basée sur un raisonnement modulo des théories. Nous définissons entre autres les concepts d'impliquants modulo une théorie, d'impliquants modulo des théories et d'hyper-impliquants. Nous proposons divers algorithmes de compilation et d'interrogation suivant ces différents types d'impliquants.

Guide de lecture

Dans le but d'éviter au lecteur de rechercher dans le document les notations, conventions et définitions nous avons essayé de regrouper au maximum celles-ci dans la première partie de la thèse.

La numérotation des définitions, théorèmes, figures, tableaux, algorithmes, etc. inclut le numéro du chapitre et l'ordre d'apparition dans ce chapitre. Cette numérotation a été préférée afin de faciliter les recherches du lecteur.

La partie I intègre un bref état de l'art sur le problème **SAT**. Volontairement cet état de l'art se veut très succinct sur les aspects algorithmiques de la résolution de **SAT**, alors que ces questions constituent l'objet principal de cette thèse. En fait, afin de toujours situer au mieux notre travail par rapport à l'existant, nous avons préféré introduire chaque thème abordé dans cette thèse par un état de l'art détaillé. Cette façon inhabituelle de procéder permettra au lecteur intéressé uniquement par un sujet précis de cette thèse, de limiter sa lecture au chapitre concernant ce sujet et d'y trouver à la fois un état de l'art et notre contribution à ce thème.

Les parties II et III étant faiblement couplées, elles peuvent donc être lues indépendamment.

L'annexe décrit différentes instances de **SAT** issues du challenge DIMACS dont il est fait référence à de multiples reprises dans ce manuscrit.

Bonne lecture ...

Première partie

Logique propositionnelle et SAT

Chapitre 1

Introduction

La logique propositionnelle, introduite sous sa forme moderne par G. Boole [Boole 1854] il y a presque 150 ans, constitue un formalisme simple et quelque peu rudimentaire. Cependant de nombreuses questions théoriques liées à cet outil de modélisation de la connaissance et du raisonnement restent ouverts et de nombreux problèmes pratiques en informatique peuvent se représenter de manière simple et se résoudre en logique propositionnelle.

Un des objectifs de cette partie introductive à nos travaux est de rappeler brièvement des fondements de la logique propositionnelle. Nous nous contentons de présenter les notations, conventions, définitions et propriétés nécessaires à la lecture de ce manuscrit. En effet, de très nombreux ouvrages (*e.g.* [Chang & Lee 1973]) traitent déjà clairement et systématiquement de ce sujet.

Cette première partie a également pour objectif de présenter les problèmes principaux de décision liés à la logique propositionnelle. Bien qu'ils s'inscrivent au sein d'un formalisme élémentaire, ces problèmes possèdent une complexité algorithmique élevée, si bien qu'en pratique il est difficile de résoudre les instances de grandes tailles.

Afin de pouvoir énoncer la complexité de ces problèmes, nous ferons quelques brefs rappels de théorie de la complexité, sans toutefois en redéfinir l'ensemble des concepts et des propriétés. Pour plus de détails quant à ce domaine, le lecteur pourra se référer par exemple aux ouvrages suivants : [Papadimitriou 1994], [Turing & Girard 1991] et [Garey & Johnson 1979].

L'ensemble de ces rappels nous permettra d'introduire le « problème de satisfaisabilité d'une formule de la logique propositionnelle mise sous forme normale conjonctive », appelé plus communément *le problème SAT*. Ce problème occupe un rôle central en théorie de la complexité, où il représente le problème NP-complet de référence [Cook 1971], auquel de nombreux problèmes en informatique se ramènent naturellement ou le contiennent dans les domaines de la programmation logique, des bases de données déductives et de la validation de circuits électroniques, à titre d'exemples. Cette thèse a pour objectif principal d'apporter de nouvelles contributions pour la résolution heuristique de grandes instances de **SAT**.

Par ailleurs, un large paragraphe sera consacré à la présentation des classes polynomiales du problème SAT. Bien que la nature de ces classes polynomiales les rende difficilement exploitables pour de très nombreuses applications, une utilisation originale et efficace de celles-ci est présentée dans la troisième partie de cette thèse. Ce chapitre introductif nous permettra donc de décrire leur construction et de présenter les algorithmes nécessaires à la reconnaissance d'instances appartenant à ces classes et à leur résolution en

temps et espace polynomial. La liste dressée dans ce chapitre des classes polynomiales de **SAT** ne se veut pas exhaustive mais constitue un simple récapitulatif des classes les plus connues et les plus utilisées.

Enfin, nous présenterons une partie des problèmes qui gravitent autour de ce test de satisfaisabilité et dont il sera fait référence dans le reste de cette thèse. Nous étudierons également leur complexité ceci afin de les situer les uns par rapport aux autres et de mieux appréhender leur difficulté.

Chapitre 2

La logique propositionnelle

2.1 Syntaxe

Définition 2.1 (atome)
Une **proposition atomique** *(ou* **atome***) est une variable booléenne, c'est-à-dire une variable qui prend des valeurs de vérité. Nous notons l'ensemble des valeurs de vérité* $\{$*FAUX, VRAI*$\}$ *ou* $\{\mathbb{F}, \mathbb{V}\}$ *ou encore* $\{0, 1\}$.

Définition 2.2 (formule)
Soit un alphabet constitué d'un ensemble fini d'atomes et de deux symboles particuliers $\top, \bot$.
Soient les connecteurs :

- *de négation :* $\neg$ *(non ...)*
- *de conjonction :* $\wedge$ *(... et ...)*
- *de disjonction :* $\vee$ *(... ou ...)*
- *d'implication :* $\rightarrow$ *(si ... alors ...)*
- *d'équivalence :* $\leftrightarrow$ *(... si et seulement si ...)*

et les opérateurs auxiliaires de parenthésage : « (» *et* «) ».
Une **formule** *propositionnelle* $\mathcal{F}$ *se construit récursivement en appliquant un nombre fini de fois les règles suivantes :*

1. *un atome,* $\top$ *et* $\bot$ *sont des formules ;*
2. *si* $\mathcal{F}$ *est une formule alors* $(\mathcal{F})$ *est une formule ;*
3. *si* $\mathcal{F}$ *est une formule alors* $\neg\mathcal{F}$ *est une formule ;*
4. *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.*

Remarque 2.1
Nous conviendrons que les connecteurs $\wedge$, $\vee, \rightarrow$ et $\leftrightarrow$ sont de même priorité, l'odre d'évaluation étant ainsi déterminé par les parenthèses. L'opérateur de négation est considéré quant à lui comme prioritaire sur tout autre connecteur.

Définition 2.3 (formules indépendantes)
Les formules $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n$ *sont dites* **indépendantes** *si et seulement si elles n'ont pas d'atomes en com-*

mun.

Définition 2.4 (littéral)
On désigne par **littéral** *un atome α ou sa négation $\neg\alpha$: α est appelé* **littéral positif** *et $\neg\alpha$* **littéral négatif**. *On dit que α et $\neg\alpha$ sont complémentaires et on note $\sim\lambda$ le* **littéral complémentaire** *du littéral λ.*

Définition 2.5 (littéral pur ou monotone)
Un littéral λ est dit **pur** *pour la formule $\mathcal{F}$ si et seulement si λ apparaît dans $\mathcal{F}$ et $\sim\lambda$ n'apparaît pas dans $\mathcal{F}$.*

2.2 Sémantique

2.2.1 Interprétation d'une formule

Définition 2.6 (interprétation)
Une **interprétation** *I en calcul propositionnel est une application de l'ensemble des formules propositionnelles dans l'ensemble des valeurs de vérité ($\{\mathbb{V}, \mathbb{F}\}$).*
L'interprétation $I(\mathcal{F})$ d'une formule $\mathcal{F}$ est définie par la valeur de vérité donnée à chacun des atomes de $\mathcal{F}$.
$I(\mathcal{F})$ est calculée par l'intermédiaire des règles suivantes :

1. $I(\top) = \mathbb{V}$;
2. $I(\bot) = \mathbb{F}$;
3. $I(\neg\mathcal{F}) = \mathbb{V}$ *si et seulement si* $I(\mathcal{F}) = \mathbb{F}$;
4. $I(\mathcal{F} \wedge \mathcal{G}) = \mathbb{V}$ *si et seulement si* $I(\mathcal{F}) = I(\mathcal{G}) = \mathbb{V}$;
5. $I(\mathcal{F} \vee \mathcal{G}) = \mathbb{F}$ *si et seulement si* $I(\mathcal{F}) = I(\mathcal{G}) = \mathbb{F}$;
6. $I(\mathcal{F} \rightarrow \mathcal{G}) = \mathbb{F}$ *si et seulement si* $I(\mathcal{F}) = \mathbb{V}$ *et* $I(\mathcal{G}) = \mathbb{F}$;
7. $I(\mathcal{F} \leftrightarrow \mathcal{G}) = \mathbb{V}$ *si et seulement si* $I(\mathcal{F}) = I(\mathcal{G})$.

La table suivante synthétise les valeurs de vérité pour les formules $(\neg\mathcal{F})$, $(\mathcal{F} \wedge \mathcal{G})$, $(\mathcal{F} \vee \mathcal{G})$, $(\mathcal{F} \rightarrow \mathcal{G})$ et $(\mathcal{F} \leftrightarrow \mathcal{G})$ en fonction de celles de $\mathcal{F}$ et $\mathcal{G}$:

$I(\mathcal{F})$	$I(\mathcal{G})$	$I(\neg\mathcal{F})$	$I(\mathcal{F} \wedge \mathcal{G})$	$I(\mathcal{F} \vee \mathcal{G})$	$I(\mathcal{F} \rightarrow \mathcal{G})$	$I(\mathcal{F} \leftrightarrow \mathcal{G})$
$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{V}$	$\mathbb{V}$

TAB. 2.1 – *Table de vérité d'une formule*

Note 2.1

1. Utilisant les valeurs 0 et 1 pour $\mathbb{F}$ et $\mathbb{V}$, il est possible d'écrire :
   - $I(\neg\mathcal{F}) = 1 - I(\mathcal{F})$;
   - $I((\mathcal{F} \wedge \mathcal{G})) = \min\{I(\mathcal{F}), I(\mathcal{G})\}$;
   - $I((\mathcal{F} \vee \mathcal{G})) = \max\{I(\mathcal{F}), I(\mathcal{G})\}$.
2. De la table de vérité ci-dessus, il est possible de déduire que :
   - la formule $(\mathcal{F} \rightarrow \mathcal{G})$ s'écrit également $(\neg\mathcal{F} \vee \mathcal{G})$;
   - la formule $(\mathcal{F} \leftrightarrow \mathcal{G})$ s'écrit également $((\mathcal{F} \rightarrow \mathcal{G}) \wedge (\mathcal{G} \rightarrow \mathcal{F}))$ et donc $((\neg\mathcal{F} \vee \mathcal{G}) \wedge (\neg\mathcal{G} \vee \mathcal{F}))$;

- le « ou exclusif » (exprimant la « différence ») de deux formules, noté : $(\neg\mathcal{F} \oplus \mathcal{G})$, s'écrit également : $\neg(\mathcal{F} \leftrightarrow \mathcal{G})$. Ou encore, si l'on se restreint à l'utilisation des opérateurs $\{\neg, \vee, \wedge\}$: $((\mathcal{F} \wedge \neg\mathcal{G}) \vee (\mathcal{G} \wedge \neg\mathcal{F}))$.

3. Nous notons $\mathbb{S}_{I(\mathcal{F})}$ l'ensemble des interprétations d'une formule $\mathcal{F}$. Si $\mathcal{F}$ possède exactement n atomes différents, alors $|\mathbb{S}_{I(\mathcal{F})}| = 2^n$.

Remarque 2.2
En calcul propositionnel, pour décrire une interprétation I, il suffit de connaître la valeur de vérité qu'elle donne à toute variable propositionnelle v de la formule, sachant que soit le littéral v, soit le littéral $\neg v$ est vrai dans I. Il est donc naturel de caractériser une interprétation I par l'ensemble des atomes vrais pour l'interprétation. Par exemple, pour la formule $(a \wedge b) \rightarrow (c \leftrightarrow d)$ et l'interprétation $I(a) = \mathbb{V}$, $I(b) = \mathbb{F}$, $I(c) = \mathbb{V}$ et $I(d) = \mathbb{F}$, I est notée $\{a, c\}$. Cependant, cette représentation ne permet pas de distinguer les atomes n'ayant pas de valeur de vérité « définie », des atomes ayant une valeur de vérité égale à $\mathbb{F}$, dans le cas des interprétations incomplètes (cf. définition 2.8). Nous adopterons donc la convention suivante consistant à représenter une interprétation par l'ensemble des littéraux vrais pour l'interprétation. Ainsi, selon cette seconde convention, l'interprétation I explicitée ci-dessus sera représentée par l'ensemble $\{a, \neg b, c, \neg d\}$.

Définition 2.7 (Distance de Hamming)
Soient I et I' deux interprétations appartenant à $\mathbb{S}_{I(\mathcal{F})}$, nous notons $\mathbb{S}_{H(I,I')}$ l'ensemble des variables pour lesquelles I et I' diffèrent : $\mathbb{S}_{H(I,I')} = \{x$ tel que x est un atome de $\mathcal{F}$ et $I(x) \neq I'(x)\}$
La **distance de** HAMMING *entre deux interprétations I et I' d'une formule $\mathcal{F}$, notée $d_H(I, I')$, représente le cardinal de cet ensemble : $d_H(I, I') = |\mathbb{S}_{H(I,I')}|$.*
Si l'on utilise la représentation ensembliste des interprétations (définie par la seconde convention proposée dans la remarque 2.2), nous avons : $d_H(I, I') = |I| - |I \cap I'|$.

Définition 2.8 (Interprétation partielle, incomplète et complète)
Soit I une interprétation (vue comme un ensemble de littéraux) d'une formule $\mathcal{F}$. On dit que :

- *I' est une* **interprétation partielle** *de $\mathcal{F}$ si et seulement si $I' \subseteq I$;*
- *I' est une* **interprétation incomplète** *de $\mathcal{F}$ si et seulement si $I' \subset I$.*

Note 2.2
Une interprétation I de $\mathcal{F}$ est donc une interprétation partielle de $\mathcal{F}$, elle est également appelée **interprétation complète**. Dans le reste du manuscrit, le terme « interprétation complète » sera préféré à celui « d'interprétation » lorsqu'il sera nécessaire d'opposer cette interprétation à une interprétation incomplète.

Remarque 2.3
Pour une formule ayant n atomes distincts, une interprétation (complète) est donc caractérisée par un ensemble de n littéraux tel que cet ensemble ne contient pas de littéraux complémentaires. Dans le cas d'une interprétation partielle (repectivement incomplète) le cardinal de cet ensemble de littéraux est inférieur ou égal (respectivement strictement inférieur) à n.

Définition 2.9 (Prolongement d'une interprétation partielle)
On appelle **prolongement** *d'une interprétation partielle I_p vers une interprétation complète I d'une formule $\mathcal{F}$, l'inteprétation partielle I'_p telle que $I_p \cup I'_p = I$.*

Note 2.3
Le prolongement I'_p peut éventuellement être l'ensemble vide.

Définition 2.10 (interprétation complémentaire ou « miroir »)
L' **interprétation complémentaire** *de l'interprétation I, notée $\sim I$ est telle que :*
$\forall \alpha$ un atome tel que $I(\alpha)$ est définie, $\sim I(\alpha) = \neg(I(\alpha))$.

En référence à la remarque 2.2, $\sim I$ est caractérisée par l'ensemble des littéraux complémentaires à ceux caractérisant I.

2.2.2 Consistance, inconsistance, validité et invalidité d'une formule

Définition 2.11 (formule satisfaite)
On dit qu'une formule propositionnelle $\mathcal{F}$ est **satisfaite** *par une interprétation I si et seulement si :* $I(\mathcal{F}) = \mathbb{V}$.

Définition 2.12 (formule falsifiée)
Dualement, on dit qu'une formule propositionnelle $\mathcal{F}$ est **falsifiée** *(ou contredite) par une interprétation I si et seulement si* $I(\mathcal{F}) = \mathbb{F}$.

Définition 2.13 (modèle)
On appelle **modèle** *d'une formule $\mathcal{F}$, une interprétation I telle que I satisfait $\mathcal{F}$.*

Définition 2.14 (contre-modèle)
On appelle **contre-modèle** *d'une formule $\mathcal{F}$, une interprétation I telle que I falsifie $\mathcal{F}$.*

Les concepts de consistance, inconsistance, validité et invalidité d'une formule se définissent comme suit.

Définition 2.15 (consistance, inconsistance)
Une formule est dite **consistante** *si et seulement si elle admet au moins un modèle.*
Une formule est dite **inconsistante** *si et seulement si elle n'admet pas de modèle, c'est-à-dire :* $\forall\ I \in \mathcal{S}_{I(\mathcal{F})},\ I(\mathcal{F}) = \mathbb{F}$.

Définition 2.16 (validité, invalidité)
Une formule est dite **valide** *(ou* **universellement valide** *ou* **tautologique***) si et seulement si elle n'admet pas de contre-modèle, c'est-à-dire* $\forall\ I \in \mathcal{S}_{I(\mathcal{F})},\ I(\mathcal{F}) = \mathbb{V}$.
Une formule est dite **invalide** *si et seulement si elle admet un contre-modèle.*

2.2.3 Conséquence sémantique

Définition 2.17 (conséquence sémantique ou conséquence logique)
Une formule $\mathcal{F}$ **implique sémantiquement** *une formule $\mathcal{G}$, noté $\mathcal{F} \models \mathcal{G}$, si et seulement si tout modèle de $\mathcal{F}$ est un modèle de $\mathcal{G}$.*
On dit alors que $\mathcal{F}$ a pour **conséquence logique** *$\mathcal{G}$, ou encore que $\mathcal{G}$ est une* **conséquence logique** *de $\mathcal{F}$.*

Définition 2.18 (littéral impliqué)
On dit qu'une littéral l est un **littéral impliqué** *d'une formule $\mathcal{F}$ si et seulement si $\mathcal{F} \models l$, c'est-à-dire si l appartient à tout modèle de $\mathcal{F}$.*

Définition 2.19 (équivalence sémantique)
On dit que deux formules $\mathcal{F}$ et $\mathcal{G}$ sont **sémantiquement équivalentes** *, noté $\mathcal{F} \equiv \mathcal{G}$, si et seulement si $\mathcal{F} \models \mathcal{G}$ et $\mathcal{G} \models \mathcal{F}$, c'est-à-dire si elles admettent exactement les mêmes modèles.*

Exemple 2.1 (Exemple d'équivalence)
Un exemple trivial d'équivalence sémantique de deux formules : $(\mathcal{F} \rightarrow \mathcal{G}) \equiv (\neg\mathcal{F} \vee \mathcal{G})$

La proposition suivante exprime un résultat fondamental en démonstration automatique (preuve par l'absurde).

Propriété 2.1
Soient deux formules $\mathcal{F}$ et $\mathcal{G}$: $\mathcal{F} \vDash \mathcal{G}$ si et seulement si la formule $(\mathcal{F} \wedge \neg\mathcal{G})$ est inconsistante.

2.3 Clauses, monômes et formes normales

2.3.1 Clauses et monômes

Définition 2.20 (clause)
On appelle **clause** *une formule constituée d'une disjonction finie de littéraux.*

Définition 2.21 (monôme)
Dualement, on appelle **monôme** *une formule constituée d'une conjonction finie de littéraux.*

Nous noterons une clause indifféremment par l'ensemble de ses littéraux $\{l_1, l_2, \ldots, l_n\}$ ou par leur disjonction $(l_1 \vee l_2 \vee \ldots \vee l_n)$. De même un monôme pourra être noté $\{l_1, l_2, \ldots, l_n\}$ ou $(l_1 \wedge l_2 \wedge \ldots \wedge l_n)$. Les ambiguïtés possibles entre clauses et monômes, dûes à la notation ensembliste, seront levées par l'utilisation de la notation disjonctive ou conjonctive.

Définition 2.22 (clause et monôme fondamentaux)
On appelle **clause fondamentale** *(resp.* **monôme fondamental***) une clause (resp. un monôme) qui ne contient pas de littéraux complémentaires.*

Note 2.4
De manière plus générale, on appelle ensemble fondamental de littéraux, un ensemble de littéraux ne contenant pas de littéraux complémentaires. En particulier, une interprétation est un ensemble fondamental de littéraux.

Remarque 2.4
Une clause qui ne serait pas fondamentale est tautologique, alors qu'une clause fondamentale est falsifiable. Inversement, un monôme non fondamental est inconsistant, alors qu'un monôme fondamental peut être satisfait.

Définition 2.23 (clause positive, négative, mixte)
On appelle **clause positive** *(resp.* **négative***) une clause qui ne contient pas de littéraux négatifs (resp. positifs).*
Une clause constituée de littéraux positifs et négatifs sera appelée **clause mixte**.

Définition 2.24 (Longueur d'une clause ou d'un monôme)
On appelle **longueur de la clause** c *(resp. du monôme m), le nombre de littéraux différents contenus dans c (resp. m). Cette longueur sera notée $l(c)$ (resp. $l(m)$).*

Définition 2.25 (clause unaire ou unitaire)
Une **clause** c **unaire** *(ou unitaire ou mono-littérale) est une clause telle que* $l(c) = 1$.

Remarque 2.5
Un littéral apparaissant dans une clause unitaire est alors appelé **littéral unitaire** ; par abus de langage, nous l'appelons également **mono-littéral**.

Définition 2.26 (clause binaire)
Une **clause binaire** c *est une clause telle que* $l(c) = 2$.

De même, une clause ternaire est de longueur de trois ; une clause quaternaire est de longueur quatre ; etc.

Définition 2.27 (clause vide)
La **clause vide**, *c'est-à-dire la clause telle que* $l(c) = 0$, *sera notée* $\{\}$ *ou* $\bot$.

Note 2.5
La clause vide est inconsistante, et elle pourra être qualifiée de positive ou de négative, mais pas de mixte.

Définition 2.28 (monôme vide)
Le **monôme vide**, *c'est-à-dire le monôme tel que* $l(m) = 0$, *sera noté* $\{\}$ *ou* $\top$.

Note 2.6
Le monôme vide symbolise la validité, il pourra être qualifié de positif ou de négatif mais pas de mixte.

Définition 2.29 (clause de Horn)
Une **clause de Horn** *est une clause qui contient au plus un littéral positif.*

Note 2.7
Dualement, une clause contenant au plus un littéral négatif sera appelée **clause reverse-Horn**.

Exemple 2.2 (Exemples de clauses de Horn)
$\{\neg a, \neg b,\ d, \neg c\}$ est une clause de Horn.
$\{a,\ b, \neg d,\ c\}$ est une clause reverse-Horn.
Les clauses négatives sont des clauses de Horn.
Les clauses positives sont des clauses reverse-Horn.

Définition 2.30 (sous-sommation ou subsumption)
La clause c_1 **sous-somme** *ou* **subsume** *la clause* c_2 *si et seulement si* $c_1 \subseteq c_2$. *Le symbole* $\subseteq$ *représentant l'inclusion ensembliste non stricte.*

Remarque 2.6
Dans le cas où c_1 et c_2 sont des clauses fondamentales, $c_1 \subseteq c_2$ si et seulement si $c_1 \vDash c_2$.

Exemple 2.3 (Exemples de subsumption)
$\{a, \neg d,\ c\}$ subsume $\{a, \neg b, \neg d,\ c\}$;
$\{a,\ b, \neg d,\ c\}$ ne subsume pas $\{\neg b,\ a,\ b\}$, bien que $\{a,\ b, \neg d,\ c\} \vDash \{\neg b,\ a,\ b\}$, mais la clause $\{\neg b,\ a,\ b\}$ n'est pas fondamentale.

Définition 2.31 (résolvante)
Deux clauses c_1 *et* c_2 **se résolvent** *si et seulement si il existe un littéral* l *tel que* $l \in c_1$ *et* $\sim l \in c_2$.
La clause $((c_1 - \{l\}) \cup (c_2 - \{\sim l\}))$ *est appelée* **résolvante** *, plus précisément résolvante en* l *de* c_1 *et* c_2.

Exemple 2.4 (Exemple de résolvante)
$\{a, \neg b, \neg d,\ c\}$ et $\{\neg b,\ d, \neg e,\ f\}$ se résolvent en d et ont pour résolvante $\{a, \neg b,\ c, \neg e,\ f\}$

2.3.2 Formes normales

Définition 2.32 (CNF)
Une formule $\mathcal{F}$ est sous **forme normale conjonctive (CNF)** *si et seulement si $\mathcal{F}$ est une conjonction de clauses, c'est-à-dire une conjonction de disjonctions de littéraux.*

Définition 2.33 (DNF)
Dualement, une formule $\mathcal{F}$ est sous **forme normale disjonctive (DNF)** *si et seulement si $\mathcal{F}$ est une disjonction de monômes, c'est-à-dire une disjonction de conjonctions de littéraux.*

Propriété 2.2
Toute formule propositionnelle peut être réécrite sous forme normale.

Cependant la transformation classique peut nécessiter une croissance exponentielle de la taille de l'ensemble obtenu. Dans [Siegel 1987] un algorithme permettant de transformer en temps linéaire toute formule $\mathcal{F}$ en une forme normale conjonctive, équivalente du point de vue de la satisfaisabilité, est donné.

Note 2.8
Une forme normale conjonctive (resp. disjonctive) est représentée par l'ensemble de ses clauses (resp. monômes) : $\{c_1, c_2, \ldots, c_n\}$ (resp. $\{m_1, m_2, \ldots, m_n\}$).

Définition 2.34 (CNF irredondante ou minimale)
Soit Σ une formule mise sous forme normale conjonctive, Σ est **irredondante minimale** *si et seulement si pour toute clause c de Σ : $(\Sigma \setminus \{c\}) \not\models c$.*

Définition 2.35 (base)
Une **base** *est un ensemble fini de formules que l'on assimile à la conjonction de ses formules.*

Note 2.9
Dans le reste du manuscrit, une base sera assimilée à une conjonction de clauses qui lui est équivalente du point de vue de la satisfaisabilité.

Nous noterons $\mathcal{S}_C$ l'ensemble des clauses d'une base et C (ou $|\mathcal{S}_C|$) son cardinal. De la même manière $\mathcal{S}_V$ et $\mathcal{S}_L$ représenteront respectivement l'ensemble des variables et l'ensemble des littéraux d'une base et leur cardinalités seront notées respectivement V et L (ou $|\mathcal{S}_V|$ et $|\mathcal{S}_L|$).

Définition 2.36 (CNF « simplifié »)
Soient une CNF Σ et un littéral l, nous notons Σ_l la CNF simplifiée en supprimant de Σ toutes les occurrences de $\sim l$ et toutes les clauses où l apparaît.
Par extension, si c est une clause fondamentale, Σ_c correspond à la simplification de Σ par tous les littéraux de la clause c ($\Sigma_c \equiv \Sigma \wedge l_1 \wedge l_2 \wedge ... \wedge l_q$ où $c = \{l_1, l_2, ..., l_q\}$).
Nous notons Σ^* la CNF simplifiée par la propagation[1] *de tous ses littéraux unitaires de Σ.*
De même, nous notons $\Sigma^\diamond$ la CNF simplifiée par la propagation de tous les littéraux purs de Σ.
Enfin, Σ^+ correspond à la CNF simplifiée par la propagation de tous les littéraux unitaires et purs de Σ.

1. La propagation unitaire correspond à une « simplification récursive » des littéraux unitaires (cf. paragraphe 7.1.1.1 page 96).

Intuitivement, la simplification d'une formule par un littéral l est une opération élémentaire consistant à exercer les règles d'évaluation d'une formule sur l'interprétation partielle I_p définie uniquement pour l'atome l ($I_p(l) = \mathbb{V}$ si l est positif, $\mathbb{F}$ sinon). Dans le cas d'une CNF, cette simplification s'effectue en deux étapes :

1. suppression des clauses satisfaites par l, c'est-à-dire les clauses c telles que $l \in c$;
2. suppression des occurrences falsifiées de l, c'est-à-dire des occurrences de $\sim l$.

Cette simplification est couramment appelée affectation (ou assignation) d'un littéral.

Exemple 2.5 (Simplifications d'une CNF)
Soit la CNF $\Sigma = \{(a \vee \neg c),\ (\neg a \vee b \vee c),\ (\neg d \vee \neg a),\ (\neg c \vee d),\ (\neg b \vee e \vee f),\ (\neg e \vee f \vee \neg g),\ (\neg f \vee \neg e)\}$.
Soit la clause $C = \{a,\ \neg d\}$.
$\Sigma_a = \{(b \vee c),\ (\neg d),\ (\neg c \vee d),\ (\neg b \vee e \vee f),\ (\neg e \vee f \vee \neg g),\ (\neg f \vee \neg e)\}$.
$\Sigma_C = \{(b \vee c),\ (\neg c),\ (\neg b \vee e \vee f),\ (\neg e \vee f \vee \neg g),\ (\neg f \vee \neg e)\}$.
$\Sigma_C^* = \{(e \vee f),\ (\neg e \vee f \vee \neg g),\ (\neg f \vee \neg e)\}$.
$\Sigma_C^+ = \{(e \vee f), (\neg f \vee \neg e)\}$.

2.4 Impliqués et impliquants

Définition 2.37 (impliqué)
Soient Σ une base et c une clause, c est un **impliqué** *de Σ si et seulement si $\Sigma \models c$.*

Définition 2.38 (impliquant)
Soient Σ une base et m un monôme, m est un **impliquant** *de Σ si et seulement si $m \models \Sigma$.*

Définition 2.39 (impliqué et impliquant premiers)
Un **impliqué** *(resp.* **impliquant***)* **premier** *d'une base est un impliqué (resp. impliquant) qui n´est pas subsumé par aucun autre, c'est-à-dire qu'il est minimal pour l'inclusion.*

Exemple 2.6 (Impliqués, impliquants, impliqués premiers et impliquants premiers)
$\Sigma = \{(a \vee \neg b \vee c),\ (\neg c \vee d),\ (\neg a \vee \neg d),\ (b \vee c)\}$
$\{\neg a \vee \neg c \vee d\}$ est un impliqué de Σ (subsumé par $\{\neg a \vee \neg c\}$, impliqué premier de Σ) ;
$\{\neg a \vee b\}$ est un impliqué premier de Σ ;
$\{\neg a \wedge b \wedge c \wedge d\}$ est un impliquant de Σ (subsumé par $\{\neg a \wedge c \wedge d\}$, impliquant premier de Σ) ;
$\{a \wedge b \wedge \neg c \wedge \neg d\}$ est un impliquant premier de Σ.

Remarque 2.7
En calcul propositionnel, les notions de modèle partiel et d'impliquant se confondent.

Définition 2.40 (couverture d'impliqués)
Une **couverture d'impliqués** *d'une base Σ est un ensemble, noté $\widehat{\Sigma}_c$, de conjonctions d'impliqués de Σ tel que $\Sigma \equiv \widehat{\Sigma}_c$.*

Définition 2.41 (couverture d'impliquants)
Une **couverture d'impliquants** *d'une base Σ est un ensemble, noté $\widehat{\Sigma}_m$, de disjonctions d'impliquants de Σ tel que $\widehat{\Sigma}_m \equiv \Sigma$.*

Définition 2.42 (couverture irredondante)
Une couverture d'impliqués $\widehat{\Sigma}_c$ *est dite* **irredondante** *si et seulement si* $\nexists\, c \in \widehat{\Sigma}_c$ *t.q.* $(\widehat{\Sigma}_c \setminus c) \equiv \Sigma$.
De la même manière, une couverture d'impliquants $\widehat{\Sigma}_m$ *est dite* **irredondante** *si et seulement si* $\nexists\, m \in \widehat{\Sigma}_m$ *t.q.* $(\widehat{\Sigma}_m \setminus m) \equiv \Sigma$.

Via la même construction, il est possible de définir une couverture (irredondante ou non) d'impliqués premiers (respectivement d'impliquants premiers), c'est-à-dire une couverture constituée de conjonctions d'impliqués premiers (respectivement de disjonctions d'impliquants premiers).

Chapitre 3

Caractérisation du problème SAT

Ce chapitre a pour objectif de présenter le problème **SAT**. Pour ce faire, nous commençons par de très brefs rappels de théorie de la complexité, ceci afin de situer au mieux **SAT** et certains cas particuliers de ce problème dans la hiérarchie polynomiale. Il est à noter que le lecteur familier de la théorie de la complexité peut passer directement au paragraphe 3.2 (page 23).

Une fois ces rappels effectués, le problème de la satisfaisabilité d'une formule propositionnelle est défini formellement. Enfin le reste de ce chapitre est consacré à la présentation de quelques classes ou restrictions polynomiales de SAT.

3.1 Classes de complexité

Ce rappel ayant simplement pour objectif de situer le problème **SAT** et ses restrictions dans la hiérarchie polynomiale, nous ne rappelons pas l'ensemble des concepts et propriétés nécessaires à la compréhension de la théorie de la complexité. Pour plus de précisions quant à ce domaine, le lecteur pourra se référer utilement aux ouvrages [Papadimitriou 1994] et [Turing & Girard 1991].

3.1.1 Préliminaires : la machine de Turing

La définition des différentes classes de complexité se basant sur un modèle de machine appelée *machine de* TURING, il est nécessaire de redéfinir les fondements de celle-ci.

Définition 3.1 (Machine de Turing)
Une **machine de** TURING *est constituée des éléments suivants :*

- *une bande de longueur infinie, découpée en cases (chaque case pouvant être vide ou contenir un symbole appartenant à un alphabet fini A) ;*
- *une tête de lecture/écriture pouvant se déplacer d'une case à la fois (on note D l'ensemble des déplacements possibles*[1]*) ;*

1. Ces déplacements sont réduits à *gauche*, *droite* et *aucun déplacement*.

– *une mémoire interne finie d'états (on note E l'ensemble des états, en particulier E est constitué d'un état initial et de un ou plusieurs états d'arrêt pouvant accepter ou rejeter la donnée d'entrée) ;*
– *une table de transitions, décrivant l'exécution du programme sur la machine.*

Un calcul sur une machine de TURING s'effectue de la manière suivante :
– la donnée d'entrée est placée sur la bande, la tête de lecture pointant sur la première entrée de la donnée ;
– le programme se déroule suivant la table de transitions et se termine lorsqu'une transition conduit à un état d'arrêt.
– le résultat peut alors être lu sur la bande.

Remarque 3.1
Il est possible que la machine n'atteigne jamais un état d'arrêt, on dit alors qu'elle boucle.

Une machine de TURING est dite déterministe si l'ensemble des transitions qui lui est associé est constitué de transitions déterministes, c'est-à-dire que la table de transition décrit une application fonctionnelle. Autrement dit, pour tout couple (e, q) défini dans la table de transitions, il est associé un triplet unique (e', q', d) où e' représente le nouvel état d'arrivée, q' le symbole à écrire sur la bande à la place de q et d le déplacement à effectuer.

A l'inverse, une machine de TURING non déterministe possède au moins une transition indéterministe dans sa table des transitions.

Les ordinateurs actuels étant conçus d'après un modèle de machine à accès aléatoire, il est donc naturel de se demander pourquoi baser nos définitions sur un modèle de calcul aussi rudimentaire que la machine de TURING. Les machines à accès aléatoire possédant une mémoire adressable ont un accès direct à l'information, contrairement à la machine de TURING qui est obligée de parcourir séquentiellement son ruban pour obtenir un résultat intermédiaire. Cependant il est possible de montrer que tout calcul nécessitant un temps en $f(n)$ sur une machine à accès aléatoire, peut être traduit sur une machine de TURING effectuant le même calcul en temps $\mathcal{O}(f^6(n))$; voire même $\mathcal{O}(f^3(n))$ avec une machine de TURING à plusieurs rubans [Papadimitriou 1994].

Propriété 3.1
Tout calcul s'effectuant en un temps $f(n)$, n représentant la taille de la donnée d'entrée, sur une machine à accès aléatoire, peut être effectué en $\mathcal{O}(f^3(n))$ sur une machine de TURING *(cf. [Papadimitriou 1994] Theorem 2.5 page 43).*

Malgré cette augmentation de temps, non négligeable, cette propriété a permis d'utiliser la machine de TURING comme référence dans les définitions théoriques des classes de complexité.

3.1.2 Classes de complexité P, NP, CoNP des problèmes de décision

Note 3.1
Dans la suite de ces rappels de théorie de la complexité, n représentera la taille de la donnée d'entrée des problèmes.

Définition 3.2 (TIME)
Un problème appartient à la classe de complexité **TIME(f(n))** *si et seulement si il existe une machine de* TURING *le résolvant en temps $\mathcal{O}(f(n))$.*

Définition 3.3 (SPACE)
Un problème appartient à la classe de complexité **SPACE(f(n))** *si et seulement si il existe une machine de* TURING *le résolvant en espace* $\mathcal{O}(f(n))$.

Définition 3.4 (Problème de décision)
Un **problème** Π **de décision** *est un problème n'ayant que deux solutions possibles :* « *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 *représentant l'ensemble des instances telles qu'il existe un programme les résolvant et répondant* « *Oui* » *;*
2. N_Π *représentant l'ensemble des instances pour lesquelles la réponse est* « *Non* ».

Y_Π *(respectivement* N_Π*) est appelé l'ensemble des* **instances positives** *(respectivement* **négatives***) du problème* Π.

Définition 3.5 (Problème de décision complémentaire)
Le problème complémentaire d'un problème de décision Π *est le problème de décision* Π^c *tel que :*

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

Définition 3.6 (P)
La classe **P** *est l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de* TURING *déterministe, soit :*

$$\mathbf{P} = \bigcup_{i \geqslant 0} \mathbf{TIME}(n^i) \tag{3.1}$$

Définition 3.7 (NP)
La classe de complexité **NP** *est l'ensemble des problèmes de décision tel que pour tout problème de* **NP***, il existe au moins une machine de* TURING *non déterministe, telle que :*

- *quels que soient les choix effectués, la machine s'arrête après un temps borné par un polynôme fonction de la taille de la donnée d'entrée ;*
- *si le problème admet une solution, la machine permet au moins une exécution s'arrêtant dans un état d'acceptation ;*
- *si le problème n'admet pas de solution, quelle que soit l'exécution, la machine ne s'arrête pas dans un état d'acceptation.*

Propriété 3.2
Clairement, nous avons : $\mathbf{P} \subseteq \mathbf{NP}$.

Définition 3.8 (CoNP)
La classe de complexité **CoNP** *est l'ensemble des problèmes de décision dont les problèmes complémentaires appartiennent à la classe* **NP**.

Propriété 3.3
De même, nous avons : $\mathbf{P} \subseteq \mathbf{CoNP}$.

Deux conjectures semblent particulièrement vraisemblables au regard de nos connaissances théoriques et pratiques actuelles [Papadimitriou 1994].

Conjecture 3.1
P $\neq$ **NP**

Conjecture 3.2
NP $\neq$ **CoNP**

3.1.3 Réductions et complétude

Définition 3.9 (Réduction)
Une fonction de réduction d'un problème Π_1 en un problème Π_2 est une fonction f_r des instances de Π_1 vers celles de Π_2 telle que :

- *f_r est calculable en espace logarithmique sur une machine de* TURING *;*
- *I est une instance positive de Π_1 si et seulement si $f_r(I)$ est une instance positive de Π_2.*

Bien que basées sur des fonctions calculables en espace logarithmique, les réductions sont couramment appelées réductions polynomiales. Ceci est du à la propriété suivante.

Propriété 3.4
Toute fonction calculable en espace logarithmique sur une machine de TURING *peut être calculée en temps polynomial (cf. [Papadimitriou 1994] Proposition 8.1 page 160).*

La notion de réduction permet de définir celle de complétude.

Définition 3.10 (Complétude)
On dit qu'un problème Π_1 est complet pour sa classe, si pour tout problème Π_2 appartenant à la même classe que Π_1, il existe une réduction de Π_2 à Π_1.

Définition 3.11 (NP-complet,CoNP-complet)
Un problème est dit **NP-complet** *s'il est complet pour la classe* **NP**.
Un problème est dit **CoNP-complet** *s'il est complet pour la classe* **CoNP**.

La conjecture 3.1 implique qu'il n'existe pas d'algorithme déterministe permettant de résoudre un problème **NP-complet** en temps polynomial dans le pire cas. La classe des problèmes **NP-complets** n'a pas seulement un intérêt théorique, en effet de nombreux problèmes pratiques, relevant notamment du monde industriel (ordonnancement, études de fiabilité, logistique, etc.), sont **NP-complets**. La figure 3.1 permet de situer les classes **P**, **NP**, **CoNP**, **NP-complet** et **CoNP-complet** les unes par rapport aux autres, sous les conjectures 3.1 et 3.2. Pour plus de détails sur la théorie de la NP-complétude, le lecteur pourra se référer à [Garey & Johnson 1979].

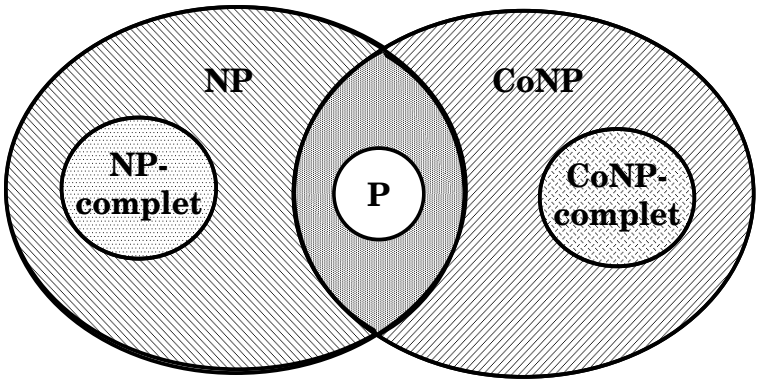

FIG. 3.1 – *Classes* **P**, **NP** *et* **CoNP**

3.2 Définition

SAT (forme abrégée de « problème de satisfaisabilité d'une formule propositionnelle sous forme normale conjonctive ») est le plus célèbre des problèmes NP-complets. Il représente un problème fondamental, à la fois théoriquement et pratiquement. En effet il occupe un rôle central en théorie de la complexité, où il représente le problème NP-complet de référence [Cook 1971], et de nombreux problèmes en informatique s'y ramènent naturellement ou le contiennent (*e.g.* démonstration automatique, programmation logique, bases de données déductives, validation de circuits électroniques, etc.).

Définition 3.12 (SAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses construites sur $\mathcal{S}$.
Question : Existe-t-il une interprétation sur $\mathcal{S}$ qui satisfasse l'ensemble des clauses de Σ ?

Définition 3.13 (kSAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses, tel que chaque clause possède au plus **k** *littéraux, construits sur $\mathcal{S}$.*
Question : Existe-t-il une interprétation sur $\mathcal{S}$ qui satisfasse l'ensemble des clauses de Σ ?

3.3 Classes polynomiales de SAT

Nous rappelons que définir une classe polynomiale nécessite deux algorithmes de complexité polynomiale :

- un algorithme de reconnaissance, qui décide si l'instance du problème appartient à cette classe ou pas ;
- un algorithme de décision de la satisfaisabilité des instances de cette classe.

Il est clair qu'il existe des classes d'instances ne répondant qu'à l'un ou l'autre des critères. Ces classes sont quasiment inexploitables en pratique. Nous pouvons citer par exemple, la classe constituée de l'ensemble des instances de **SAT** ayant un nombre polynomial de résolvantes. Clairement cette classe admet un algorithme de décision polynomial [2]. Nous appellons *restriction polynomiale* une classe pour laquelle seul un algorithme de décision polynomial existe.

Définition 3.14 (Restriction polynomiale)
On appelle **restriction polynomiale** *d'un problème Π, la restriction de Π à un ensemble d'instances de Π, tel que cet ensemble appartienne à la classe de complexité* **P**.

Définition 3.15 (Classe polynomiale ou traitable)
On appelle **classe polynomiale** *ou* **classe traitable** *d'un problème Π, une restriction polynomiale de Π telle qu'il existe un algorithme décidant en temps et en espace polynomial si une instance de Π appartient ou n'appartient pas à la classe.*

Très peu de classes polynomiales de **SAT** sont connues à ce jour. Nous en établissons dans ce paragraphe une liste non exhaustive.

2. Il suffit de calculer l'ensemble de ces résolvantes, ce qui est fait en temps et en espace polynomial par définition de la classe, et de répondre « oui l'instance admet une solution » ou « non l'instance n'admet pas de solution », en fonction de la production ou non de la clause vide.

3.3.1 Les clauses binaires

La première classe polynomiale à avoir été exhibée est *2***SAT** [Cook 1971], c'est-à-dire les instances de **SAT** formées de clauses binaires. L'algorithme de reconnaissance pour cette classe est trivial et clairement linéaire. L'algorithme de décision de satisfaisabilité proposé par S.A. Cook se base sur le principe de résolution et est assurément polynomial puisque la résolvante de deux clauses binaires est au pire une clause binaire et que le nombre de clauses binaires pouvant être construites, pour un nombre donné de variables, est une fonction quadratique de ce nombre de variables.

Propriété 3.5
*2***SAT** $\in$ **P** *[Cook 1971].*

Par la suite d'autres algorithmes de complexité linéaire furent proposés. Citons celui de S. Even, A. Itai et A. Shamir [Even *et al.* 1976] se basant sur un algorithme énumératif avec un principe de retour arrière intelligent, ou encore celui de B. Aspvall, M. Plass et R. Tarjan [Aspvall *et al.* 1979] utilisant un codage des clauses en termes de graphes et l'algorithme de R. Tarjan [Tarjan 1972] pour calculer les composantes fortement connexes de ce graphe.

3.3.2 Les clauses de Horn

Une autre classe polynomiale bien connue de **SAT** est **HORN-SAT**.

Définition 3.16 (HORN-SAT)
HORN-SAT *est la restriction de* **SAT** *aux ensembles de clauses de Horn.*

L'algorithme de reconnaissance, comme pour les clauses binaires, est trivial et clairement linéaire : il suffit de passer en revue les clauses une à une et de s'assurer qu'il y a bien au plus un littéral positif par clause. En ce qui concerne le test de satisfaisabilité d'un ensemble de clauses de Horn, plusieurs auteurs ont montré que la résolution des clauses positives unitaires, qui s'effectue en temps et en espace linéaire, est suffisante [Minoux 1988, Dalal 1992, Rauzy 1995b]. D'autres algorithmes de complexité linéaire, se basant sur la théorie des graphes, ont été proposés [Dowling & Gallier 1984], [Ghallab & Escalada-Imaz 1991] et [Scutellà 1990].

Propriété 3.6
HORN-SAT $\in$ **P**.

Définition 3.17 (Renommages)
On appelle **renommage d'un ensemble de littéraux** $\mathcal{S}_L$, *une application* ρ *telle que :*

$$\rho : \quad \mathcal{S}_L \quad \longmapsto \quad \mathcal{S}_L : \qquad l \quad \longrightarrow \quad \rho(l) = \begin{cases} \neg l \\ \text{ou} \\ l \end{cases} \qquad \text{et} \qquad \forall\, l \in \mathcal{S}_L : \rho(\neg l) = \neg\rho(l).$$

Par extension, nous définissons un **renommage** ρ_c **d'une clause** *c comme l'application de* ρ *à tous les littéraux de cette clause :* $\rho_c(c) = \bigvee_{l \in c} \rho(l)$.

De même, un **renommage** ρ_Σ **d'un ensemble de clauses** Σ *se définit comme :* $\rho_\Sigma(\Sigma) = \bigwedge_{c \in \Sigma} \rho_c(c)$.

Définition 3.18 (formule Horn-renommable)
On dit qu'un ensemble de clauses Σ est **Horn-renommable** *si et seulement si il existe un renommage ρ des littéraux de Σ tel que $\rho_\Sigma(\Sigma)$ soit un ensemble de clauses de Horn.*

Exemple 3.1 (Une instance Horn-renommable)
Soient $\Sigma = \{a \vee b \vee \neg d,\ \neg b \vee \neg c,\ a \vee d \vee \neg c,\ c \vee d\}$ et ρ le renommage suivant :

$$\begin{array}{rcl} \rho(a) & = & \neg a \\ \rho(b) & = & \neg b \\ \rho(c) & = & c \\ \rho(d) & = & \neg d \end{array}$$

Nous obtenons : $\rho_\Sigma(\Sigma) = \{\ \neg a \vee \neg b \vee d,\ b \vee \neg c,\ \neg a \vee \neg d \vee \neg c,\ c \vee \neg d\}$ qui est un ensemble de clauses de Horn. Par conséquent Σ est bien une formule Horn-renommable. □

Le problème de reconnaissance de formules Horn-renommables n'est pas aussi trivial que celui des clauses de Horn ou des clauses binaires. En effet, il n'est pas immédiat de déterminer si une instance **SAT** est de Horn à un renommage près des littéraux. Toutefois, ce problème peut être ramené à celui de la satisfaisabilité d'un ensemble de clauses binaires, et par conséquent traité de manière linéaire [Lewis 1978, Aspvall 1980, Chandru *et al.* 1990, Hébrard 1994]. Ces algorithmes fournissant pour la plupart le renommage permettant de transformer l'ensemble de clauses en un ensemble de clauses de Horn, décider de la satisfaisabilité d'une formule Horn-renommable peut être fait en temps linéaire par l'intermédiaire d'un algorithme décisionel emprunté à la classe **HORN-SAT** .

Propriété 3.7
Les instances Horn-renommables de **SAT** *constituent une classe polynomiale de* **SAT** .

3.3.3 Généralisation de HORN-SAT et *2*SAT

Plusieurs auteurs ont cherché à généraliser les deux classes polynomiales « primaires » de **SAT** (**HORN-SAT** et *2***SAT**). Ces recherches ont donné naissance à deux types de généralisation : les clauses q-Horn et un principe de hiérarchisation des clauses.

3.3.3.1 Les clauses q-Horn

La classe des clauses q-Horn a été introduite par E. Boros *et al.* dans [Boros *et al.* 1990] et se définit de la manière suivante :

Définition 3.19 (Ensemble de clauses q-Horn)
Un ensemble de clauses Σ est dit **q-Horn** *si et seulement si il existe une partition des variables (modulo un renommage) de Σ en deux sous-ensembles disjoints H et Q tels qu'aucune clause de Σ :*

- *ne contient plus de deux variables de Q ;*
- *ne contient plus d'un littéral positif de H ;*
- *contenant un littéral positif de H ne contient de littéral positif de Q.*

La difficulté de cette classe est la reconnaissance des ensembles H et Q. En effet, une fois ces ensembles déterminés, le test de satisfaisabilité revient à affecter à la valeur $\mathbb{F}$ toutes les variables de H et à résoudre

le problème restant qui appartient clairement à *2***SAT**. En ce qui concerne l'appartenance d'une formule à la classe des clauses q-Horn, E. Boros *et al.* dans [Boros *et al.* 1994] proposent un algorithme linéaire pour déterminer les ensembles Q et H.

Cette classe qui généralise les classes **HORN-SAT** et *2***SAT** a elle même été généralisée par J.J. Hébrard dans [Hébrard 1995] où aucune condition n'est imposée à l'ensemble Q. De plus, cet auteur propose un algorithme permettant de calculer en temps linéaire le plus grand sous-ensemble H, qu'il appelle la base de Horn.

3.3.3.2 Les différentes hiérarchies

Une autre manière de généraliser les clauses de Horn passe par l'introduction de différentes formes de hiérarchie d'instances **SAT** .

Généralisation de S. Yamasaki et S. Doshita : Les premiers à avoir travaillé dans cette voie furent S. Yamasaki et S. Doshita qui dans [Yamasaki & Doshita 1983] autorisent les clauses de Horn à posséder plusieurs littéraux positifs, sous condition que ces littéraux soient en quelque sorte « imbriqués ». Ceci afin de garantir que la résolution unitaire reste suffisante pour répondre à la question de la satisfaisabilité de la formule.

V. Arvind et S. Biswas ont proposé un algorithme quadratique pour décider de la satisfaisabilité des instances de cette classe [Arvind & Biswas 1987].

Hiérarchie de G. Gallo et M.G. Scutellà : G. Gallo et M.G. Scutellà ont par la suite généralisé les travaux de S. Yamasaki et S. Doshita. Dans [Gallo & Scutellá 1988], ils définissent une hiérarchie $\Gamma_0, \Gamma_1, \ldots$ d'instances de **SAT** . Cette hérarchie se construit de la façon suivante :

Définition 3.20 (La hiérarchie de Gallo-Scutellà)
- Γ_0 = **HORN-SAT** *;*
- $\Sigma \in \Gamma_k$ *si et seulement si il existe un littéral p tel que* $\Sigma_p \in \Gamma_{k-1}$ *et* $\Sigma_{\sim p} \in \Gamma_k$.

Les instances appartenant à Γ_1 représentent en fait la classe de S. Yamasaki et S. Doshita décrite dans [Yamasaki & Doshita 1983]. G. Gallo et M.G. Scutellà ont proposé un algorithme pour reconnaître et résoudre les instances de la classe Γ_k. Cet algorithme est de complexité $\mathcal{O}(|\Sigma| n^k)$ en temps et en espace. De plus, cette classe vérifie les propriétés suivantes :

Propriété 3.8
- Γ_0 = **HORN-SAT** *;*
- $\Gamma_k \supseteq \Gamma_{k-1}, \quad k = 1,\ 2,\ \ldots$ *;*
- **SAT** $= \bigcup_{k=0}^{\infty} \Gamma_k$.

Hiérarchie de M. Dalal et D.W. Etherington : La hiérarchie de G.Gallo et M.G. Scutellà a elle même été généralisée par M. Dalal et D.W. Etherington dans [Dalal & Etherington 1992] où deux hiérarchies entrelacées (Δ_i) et (Ω_i) sont définies.

Définition 3.21 (Les hiérarchies de Dalal-Etherington)
Soit Σ une formule CNF, nous avons :

- $\Sigma \in \Delta_0$ *si et seulement si* Σ^+ *contient :*
  - *soit la clause vide ;*
  - *soit uniquement des clauses positives ;*
  - *soit uniquement des clauses négatives ;*
- $\forall\, k,\ \Sigma \in \Omega_k$ *si et seulement si*
  - $\Sigma \in \Delta_k$ *;*
  - *ou pour tout littéral p de Σ^+ :*
    - *soit* $\Sigma_p^+ \in \Delta_k$ *et* $\Sigma_{\sim p}^+ \in \Omega_k$ *;*
    - *soit* $\Sigma_p^+ \subset \Sigma$ *et* $\Sigma_p^+ \in \Omega_k$ *;*
- $\forall\, k > 0, \Sigma \in \Delta_k$ *si et seulement si il existe un littéral p de Σ^+ tel que :*
  - *soit* $\Sigma_p^+ \in \Omega_{k-1}$ *et* $\Sigma_{\sim p}^+ \in \Delta_k$ *;*
  - *soit* $\Sigma_p^+ \subset \Sigma$ *et* $\Sigma_p^+ \in \Delta_k$.

Il est à noter que contrairement à la hiérarchie de G. Gallo et M.G. Scutellà, nous avons : $2\mathbf{SAT} \subset \Omega_0$.

M. Dalal et D.W. Etherington proposent un algorithme de reconnaissance et de résolution de complexité, spatiale et temporelle $\mathcal{O}(|\Sigma|n^k)$.

Remarque 3.2
Il existe d'autres généralisations des clauses de Horn, come celles s'appuyant sur des résultats de programmation linéaire $0/1$. V. Chandru et J.N. Hooker ont défini dans [Chandru & Hooker 1990] une extension des clauses de Horn pouvant être décidée par résolution unitaire. Ces travaux furent étendus dans [Schlipf *et al.* 1995] où un algorithme pour un ensemble de formules incluant celle de V. Chandru et J.N. Hooker et celles des formules bien imbriquées (cf. paragraphe 3.3.5) définies dans [Conforti & Cornuéjols 1992] fut proposé. Le problème est que nous ne disposons pas d'algorithme de reconnaissance polynomial pour ces restrictions polynomiales, ce qui les rend inutilisables en pratique.

3.3.4 Restriction sur le nombre d'occurrences des variables

Une autre manière de forcer une instance **SAT** à être polynomiale est de resteindre le nombre d'apparitions des variables dans la formule. En particulier, C.A. Tovey a montré dans [Tovey 1984] que la classe des instances où les variables apparaissent au plus deux fois est une classe polynomiale de **SAT** .

Dans le même article C.A. Tovey a prouvé que les instances de SAT où chaque variable apparaît au plus r fois – r étant la taille de la plus longue clause de l'instance – étaient satisfaisables ; et donc sont décidables en temps et en espace polynomial.

Ce résultat fut amélioré en premier lieu par O. Dubois dans [Dubois 1990], puis par J. Kratochvil *et al.* qui montrèrent que les instances de SAT telles que le nombre d'occurrences des variables est inférieur ou égal à une fonction exponentielle de la taille maximale des clauses appartiennent à **P**.

Ces restrictions de **SAT** sur le nombre d'occurrences des variables en fonction de la taille des clauses appartiennent à un problème plus général, le problème r, s-**SAT**. Nous présentons ce problème en détail dans le prochain chapitre (cf. paragraphe 4.2.1 page 35).

3.3.5 Les formules bien imbriquées

Reprenant les travaux de David Lichtenstein [Lichtenstein 1982] sur les formules dont le graphe sous-jacent[3] est planaire, D.E. Knuth, dans [Knuth 1990], a défini la classe des formules bien imbriquées :

Définition 3.22 (Formules bien imbriquées)
Soit $\prec$ un ordre total sur les variables étendu en un préordre sur les littéraux de telle manière que l et $\sim l$ possèdent le même rang pour ce préordre.
Une clause c_1 chevauche une clause c_2 si : $\exists\ l_1 \in c_1$, $\exists\ l_2 \in c_1$ et $\exists\ l_3 \in c_2$ tels que : $l_1 \prec l_3 \prec l_2$.
Un ensemble de clauses est **bien imbriqué** *si et seulement si il ne contient pas deux clauses se chevauchant.*

D.E. Knuth n'ayant pas traité le problème de la reconnaissance de ce type de formules, P. Rossa dans [Rossa 1993], a fourni un algorithme linéaire qui reconnait une formule bien imbriquée, pour un ordre fixé des variables.

Un algorithme de décision linéaire a été proposé par D.E. Knuth dans [Knuth 1990]. Cet algorithme est toutefois, comme pour l'algorithme de reconnaissance, linéaire pour un ordre fixé sur les variables.

Nous ne disposons pas à l'heure actuelle d'algorithmes « complets » (c'est-à-dire pour tout ordre sur les variables) qui soient polynomiaux. L'intérêt de cette classe pour un bon nombre d'applications en est par conséquent très amoindri.

Exemple 3.2 (Exemple de graphe sous-jacent à une formule CNF)

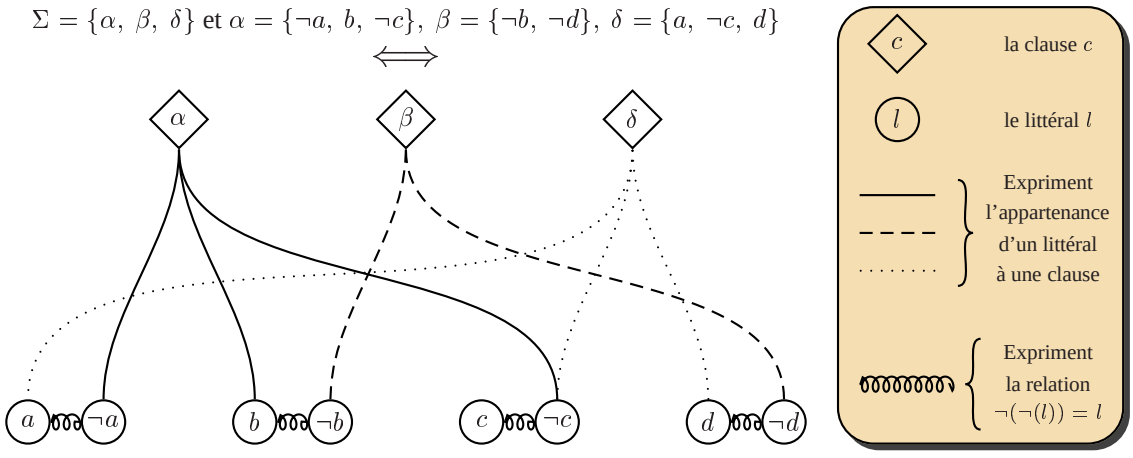

FIG. 3.2 – *Graphe sous-jacent à une formule CNF*

3. Le graphe sous-jacent d'une formule est le graphe biparti non orienté où les sommets symbolisent d'un coté les clauses et de l'autre les littéraux de la formule, tandis que les arêtes représentent l'appartenance d'un littéral à une clause. La figure ci-dessus (cf. exemple 3.2, FIG. 3.2) illustre cette représentation graphique d'une formule CNF.

3.3.6 La classe Quad

M. Dalal dans [Dalal 1996] propose une nouvelle classe polynomiale *Quad* pour **SAT** , qui peut être vue comme une extension des hiérarchies de Dalal-Etherington (cf. définition 3.21). Étant donné un ordre total $\prec$ sur les littéraux ($\prec$ sera étendu aux clauses), *Quad*, se définit récursivement de la façon suivante :

Définition 3.23 (Quad)
Un formule Σ appartient à la classe **Quad** *si et seulement si :*

1. *Σ^* la formule obtenue à partir de Σ après la propagation des clauses unitaires respecte une des quatre conditions suivantes :*
   (a) *contient la clause vide ;*
   (b) *ne contient pas de clauses positives ;*
   (c) *ne contient pas de clauses négatives ;*
   (d) *ne contient que des clauses binaires ;*
2. *ou, pour la première sous-clause maximale*[4] *μ de la première clause σ de Σ^* pour lequel $\Sigma^*_{\neg\mu}$ satisfasse la condition 1., on ait :*
   (a) *soit $\Sigma^*_{\neg\mu}$ est satisfaisable ;*
   (b) *soit la formule $\Sigma - \{\sigma\} \cup \{\mu\}$ appartient à Quad.*

Afin de mieux se représenter la classe *Quad*, nous l'illustrons sur le petit exemple proposé par M. Dalal dans [Dalal 1996].

Exemple 3.3 (Une formule appartenant à la classe Quad)
Soient $\Sigma = \{\{p,\ q,\ r\},\ \{p,\ q,\ \neg r\},\ \{\neg p,\ \neg q,\ \neg r\}\}$ et l'ordre $\prec$ induit par $p \prec q \prec r \prec \neg p \prec \neg q \prec \neg r$.

$\Sigma^* = \Sigma$ ne satisfait pas la condition 1. Soit $\sigma = \{p,\ q,\ r\}$ la première clause de Σ, il en découle que la première sous-clause maximale est égale à : $\mu = \{p,\ q\}$.

$\Sigma^*_{\neg\mu}$ contient la clause vide, par conséquent $\Sigma^*_{\neg\mu}$ satisfait bien la condition 1 mais pas la condition 2a. Il faut donc vérifier si la formule $\Sigma - \{\sigma\} \cup \{\mu\} = \{\{p,\ q\},\ \{p,\ q,\ \neg r\},\ \{\neg p,\ \neg q,\ \neg r\}\}$ appartient à la classe *Quad*. Nous noterons cette nouvelle formule Σ'.

Nous considérons donc maintenant la formule $\Sigma' = \{\{p,\ q\},\ \{p,\ q,\ \neg r\},\ \{\neg p,\ \neg q,\ \neg r\}\}$. $(\Sigma')^* = \Sigma'$, Σ' ne satisfaisant toujours pas la première condition, il nous faut donc vérifier la seconde. A cette fin, la première clause devient $\sigma' = \{p,\ q\}$ et sa première sous-clause maximale $\mu' = \{p\}$. Ainsi $\Sigma'^*_{\neg\mu'} = \{\}$, ce qui satisfait la condition 2a. Il en découle que Σ' appartient à la classe *Quad* et que par conséquent Σ appartient également à la classe *Quad*. □

Malgré une définition assez lourde, cette classe est reconnaissable en temps et en espace polynomial. En effet, M. Dalal fournit dans [Dalal 1996] un même algorithme (*QuadSat*[5]) quadratique pour la reconnaissance et la résolution de cette classe.

Cependant cette classe souffre de deux problèmes majeurs qui la rendent quasiment inexploitable. Comme pour la classe des formules bien imbriquées (cf. paragraphe 3.3.5), le premier obstacle est la détermination de l'ordre $\prec$. En effet, une formule appartient à la classe *Quad* pour un certain ordre fixé à

4. On appelle sous-clause maximale de σ une clause μ telle que μ sous-somme σ et que $|\sigma| - |\mu| = 1$. L'ordre $\prec$ permet ainsi de définir la notion de première sous-clause maximale.

5. *QuadSat* est en fait en $\mathcal{O}(n^2 k)$ où n représente la taille de la formule et k la taille de la plus longue clause de cette formule.

l'avance. Le problème de l'existence d'un tel ordre n'est décidable, dans le pire cas, qu'après l'essai des $n!$[6] ordres possibles. Ce qui rend l'algorithme de reconnaissance de la classe *Quad* pour un ordre non fixé exponentiel. Le second problème est que la classe *Quad* n'est pas stable par adjonction de clauses unitaires, ce qui rend son utilisation impossible dans de nombreuses applications (*e.g.* l'implication de clauses). Nous proposons un exemple pour illustrer cette non propriété [Marquis 1997].

Exemple 3.4 (Quad : une classe polynomiale non stable pour l'adjonction de clauses unitaires)
Soit $\Sigma = \{\{\neg x,\ \alpha_1,\ \beta_1,\ \gamma_1\},\ \{\neg x,\ \alpha_2,\ \beta_2,\ \gamma_2\},\ \{\neg\alpha_3,\ \neg\beta_3,\ \neg\gamma_3\},\ \{\neg\alpha_4,\ \neg\beta_4,\ \neg\gamma_4\},\ \{x,\ y\}\}$. Nous avons : $\forall\ \prec,\ \Sigma \in Quad$. En effet, quel que soit l'ordre choisi, nous serons amenés à considérer la sous-clause maximale $\mu = \{x\}$ car pour tous les autres μ possibles, nous avons $\Sigma^*_{\neg\mu}$ qui ne satisfait pas la condition 1 (nécessaire à la satisfaction de la condition 2). De plus nous avons : $\Sigma^*_{\neg\{x\}} = \{\{\neg\alpha_3,\ \neg\beta_3,\ \neg\gamma_3\},\ \{\neg\alpha_4,\ \neg\beta_4,\ \neg\gamma_4\}\}$ qui satisfait les conditions 1 et 2a. Par conséquent $\Sigma \in Quad$.

En revanche, qu'en est-il de $\Sigma' = \Sigma \cup \{x\}$?
Nous avons $\Sigma'^* = \{\{\alpha_1,\ \beta_1,\ \gamma_1\},\ \{\alpha_2,\ \beta_2,\ \gamma_2\},\ \{\neg\alpha_3,\ \neg\beta_3,\ \neg\gamma_3\},\ \{\neg\alpha_4,\ \neg\beta_4,\ \neg\gamma_4\}\}$. Très rapidement, il est facile de voir que quel que soit l'ordre choisi, il n'existe pas de clause μ' tel que $\Sigma'^*_{\neg\mu'}$ satisfasse la condition 1 nécessaire à la satisfaction de la condition 2.

Par conséquent nous avons $\Sigma' \notin Quad$, ce qui implique que la classe *Quad* n'est pas stable par adjonction de clauses unitaires. □

3.4 Discussion

Le problème **SAT**, malgré sa nature très élémentaire, se trouve être un problème complexe à résoudre. Étant le problème **NP-complet** de référence, les résultats théoriques, mais également pratiques, obtenus quant à la résolution de ce problème permettent indubitablement une avancée dans la résolution des nombreux problèmes qui s'y ramènent naturellement.

Il reste que la plupart des résultats obtenus sur les classes polynomiales de **SAT** s'appuient très souvent sur une restriction du langage. Cette perte d'expressivité est souvent trop contraignante pour permettre une exploitation de ces classes polynomiales[7]

Cependant, la décision de ne pas restreindre le langage s'accompagne irrémédiablement d'autres formes de restrictions comme l'utilisation d'algorithmes incomplets. En effet, pour résoudre pratiquement le problème **SAT** on utilise deux types de méthodes :

1. les méthodes incomplètes qui permettent de prouver la satisfaisabilité d'instances de « grande » taille, mais qui sont incapables de prouver l'inconsistance des instances quelle que soit leur taille ;
2. les méthodes complètes qui prouvent aussi bien la consistance que l'inconsistance, mais qui ne permettent la résolution que d'instances de taille limitée.

Ce sont ces formes de restrictions que nous avons choisies pour traiter **SAT**. Nous présentons dans la deuxième partie de ce manuscrit notre contribution aux diverses méthodes complètes et incomplètes de résolution du problème **SAT**.

Par ailleurs, dans la troisième partie de cette thèse nous proposons une exploitation originale et efficace

6. n représente ici le nombre de littéraux de la formule.
7. Il reste que les classes polynomiales peuvent être utilisées pour réaliser des « filtrages » au sein d'algorithmes (*e.g.* [Gallo & Urbani 1989]).

des classes polynomiales de **SAT** dans un contexte de compilation. Nous utilisons les classes polynomiales de **SAT** afin de diminuer au maximum la taille des compilations et sans pour autant restreindre le langage.

Chapitre 4

Les problèmes autour de SAT

Ce chapitre a pour objectif de présenter différents problèmes qui gravitent autour du test de satisfaisabilité d'un ensemble de clauses. Nous étudierons les problèmes de recherche et d'optimisation associés à **SAT**. Une large part du précédent chapitre était dédiée à la présentation des classes polynomiales de **SAT**, ce présent chapitre nous permettra d'énoncer quelques restrictions de **SAT** ne permettant pas de diminuer la complexité. Enfin, ce chapitre nous permettra d'introduire divers problèmes étroitement liés au problème **SAT** et auxquels il pourra être fait référence dans le reste de ce manuscrit.

4.1 Problèmes de recherche et d'optimisation associés à SAT

4.1.1 FSAT

FSAT est le problème de recherche associé à **SAT**, il se définit de la manière suivante :

Définition 4.1 (FSAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses construites sur $\mathcal{S}$.
Question : Si Σ est consistante alors trouver un modèle de Σ.

Beaucoup de problèmes se formalisent très facilement en terme de recherche d'un modèle d'une formule. De plus, un grand nombre de méthodes pour résoudre **SAT** exhibent un modèle pour répondre favorablement à l'existence de celui-ci. Cependant, ceci n'est pas une nécessité, la résolution, par exemple, ne cherche pas à trouver un modèle pour décider si l'instance de **SAT** est consistante ou non.

Par ailleurs, il est parfois nécessaire de connaître l'ensemble des modèles d'une formule.

Définition 4.2 (FSAT-all)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses construites sur $\mathcal{S}$.
Question : Si Σ est consistante alors déterminer tous les modèles de Σ.

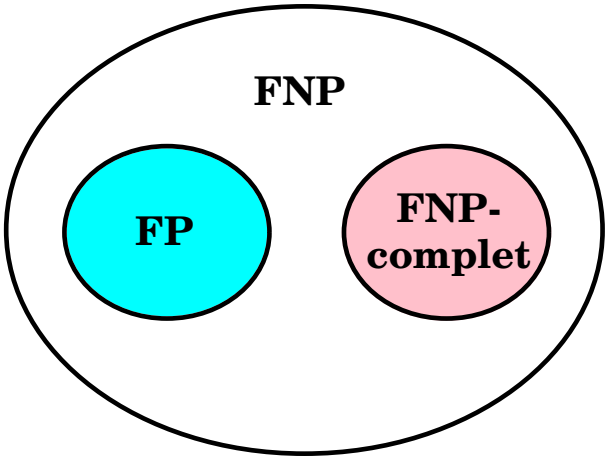

FIG. 4.1 – *Les classes* **FP**, **FNP** *et* **FNP-complet**

Il faut noter que l'ensemble des modèles d'une formule propositionnelle caractérise cette formule. En effet, si Σ possède n modèles distincts, notés m_i ($i \in [1..n]$), alors $\Sigma \equiv (m_1 \vee m_2 \vee \ldots \vee m_n)$.

Propriété 4.1
FSAT $\in$ **FNP-complet**.

Rappelons que la classe **FNP** caractérise l'ensemble des problèmes de recherche non déterministes polynomiaux, la classe **FP** caractérise les problèmes de recherche de complexité en temps polynomiale et que les classes **FP-complet** et **FNP-complet** se définissent par un concept de réduction adapté aux problèmes de recherche. La figure ci-dessus (cf. FIG. 4.1) permet de situer les classes **FP**, **FNP** et **FNP-complet** les unes par rapport aux autres, sous les conjectures 3.1 et 3.2 (cf. page 21). De plus nous avons :

Théorème 4.1
FP = **FNP** *si et seulement si* **P** = **NP** *(cf. Theorem 10.2 page 230 [Papadimitriou 1994]).*

4.1.2 #SAT

Le problème de dénombrement associé à **SAT**, se nomme **#SAT**, et peut être défini de la façon suivante :

Définition 4.3 (#SAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses construites sur $\mathcal{S}$.
Question : Combien Σ admet-il de modèles ?

Propriété 4.2
#SAT $\in$ **#P-complet**.

Rappelons que la classe de complexité **#P** représente la classe des problèmes de dénombrement associés aux problèmes de recherche de la classe **FNP**. La classe **#P-complet** se définit par un concept de réduction

adapté aux problèmes de dénombrement.

Des restrictions à ce problème de dénombrement ont été étudiées ceci afin d'en diminuer la complexité. Les premières restrictions (et les plus connues) furent :

Définition 4.4 (#rSAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses c construites sur $\mathcal{S}$ telles que $\forall c \in \Sigma,\ l(c) \leqslant r$.
Question : Combien Σ admet-il de modèles ?

Définition 4.5 (#rSAT-monotone)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses positives c construites sur $\mathcal{S}$ telles que $\forall c \in \Sigma,\ l(c) \leqslant r$.
Question : Combien Σ admet-il de modèles ?

Propriété 4.3
#rSAT *et* **#rSAT-monotone** $\in$ **#P-complet**, *pour* $r \geqslant 2$.

Il faut prendre toute la mesure de ce résultat. En effet, il montre que certains problèmes de dénombrement associés à des problèmes de décision appartenant à la classe de complexité **P**, tels que *2***SAT** et même *2***SAT-monotone**, sont **NP-difficiles**. Cependant il existe des classes polynomiales pour ce problème de dénombrement, mais elles sont extrêmement restrictives. Nous pouvons citer par exemple **#2-2SAT-monotone** les instances de SAT constituées de clauses de longueur inférieure ou égale à 2, ne contenant que des littéraux positifs et où chaque variable apparaît au plus deux fois dans l'instance !

Propriété 4.4
#2-2SAT-monotone $\in$ **#P**.*[Roth 1996]*

Dan Roth dans [Roth 1996] a prouvé qu'il existait d'autres classes polynomiales pour le problème de dénombrement, cependant elles sont souvent trop restrictives pour pouvoir trouver un cadre applicatif.

4.2 Les problèmes de décision associés à SAT

Nous ne reviendrons pas dans ce paragraphe aux restrictions polynomiales de **SAT**, tels que *2***SAT** et **HORN-SAT**. Ces problèmes ont été présentés aux paragraphes 3.3.1 et 3.3.2. Nous rappelons simplement que : *k***SAT** $\in$ **NP-complet** si et seulement si $k \geqslant 3$ et que **HORN-SAT** $\in$ **P**. Cependant **SAT** restreint aux instances comportant des clauses de Horn et des clauses binaires est **NP-complet**.

4.2.1 r, s-SAT

Différents auteurs se sont intéressés à la nature d'une instance **SAT**, si l'on restreint le nombre d'apparitions des littéraux de cette instance. Ce problème est connu sous le nom de r, s-**SAT** et défini comme suit :

Définition 4.6 (r, s-SAT)
Instance : $\mathcal{S}$ un ensemble fini de symboles propositionnels et Σ un ensemble fini de clauses construites sur $\mathcal{S}$, possédant exactement **r** *littéraux distincts, et où chaque variable de $\mathcal{S}$ apparaît au plus* **s** *fois dans Σ.*
Question : Existe-t-il une interprétation sur $\mathcal{S}$ qui satisfasse l'ensemble des clauses de Σ.

L'objectif majeur de cette restriction sur le nombre d'apparitions des littéraux était, bien évidement, d'exhiber de nouvelles instances polynomiales pour **SAT**. Reprenant cette idée, C.A. Tovey a montré dans [Tovey 1984] que la classe des instances où les variables apparaissent au plus deux fois est une classe polynomiale de **SAT** .

Propriété 4.5
$\star, 2$-**SAT**[1] $\in$ **P** *[Tovey 1984].*

Dans le même article, C.A. Tovey, utilisant un théorème de P. Hall montré dans [Hall 1935], a également prouvé que :

Propriété 4.6
$\forall r : r, r$-**SAT** *est satisfaisable. [Tovey 1984]*

C.A. Tovey avait également émis l'hypothèse que toute instance r, s-**SAT** tel que $s \leqslant 2^{r-1} - 1$ est satisfaisable. Cette conjecture fut réfutée par O. Dubois dans [Dubois 1990] où une instance incohérente **4,6-SAT** fut exhibée. Dans ce même papier, O. Dubois a établi que si toute instance r, s-**SAT** est satisfaisable alors : $\forall \lambda \in \mathbb{N}^*$ toute instance de la forme $(r + \lambda), (s + \lambda[s/r])$-**SAT** est satisfaisable[2].

Ce résultat fut amélioré par J. Kratochvil, P. Savicky et Z. Tuza dans [Kratochvtil *et al.* 1993] qui montrèrent qu'il existe un entier $f(r)$ tel que :

Propriété 4.7
Pour $s \leqslant f(r)$ r, s-**SAT** $\in$ **P**.
Pour $s > f(r)$ r, s-**SAT** $\in$ **NP-complet**.

De plus, $f(r)$ croît exponentiellement avec r de telle façon que : $[2^r/e^r] \leqslant f(r) \leqslant 2^{r-1} - 2^{r-4} - 1$.

Enfin, il faut noter qu'il existe d'autres restrictions portant essentiellement sur le nombre de clauses pouvant partager les mêmes variables [Humbert 1995]. Ces restrictions restent malheureusement **NP-complètes**.

4.2.2 MAXSAT

Bien qu'énoncé le plus souvent dans sa forme décisionnelle, **MAXSAT** peut être vu comme le problème d'optimisation de **SAT** .

Définition 4.7 (MAXSAT)
Instance : Soient $\mathcal{S}$ un ensemble fini de symboles propositionnels, Σ un ensemble fini de clauses construites sur $\mathcal{S}$

1. $\star$ signifiant pour tout *r*.
2. $[s/r]$ désigne la partie entière de (s/r).

et k un entier naturel.
Question : Existe-t-il une interprétation de Σ qui satisfasse au moins k clauses de Σ.

Plus généralement, nous avons :

Définition 4.8 (MAXrSAT)
Instance : Soient $\mathcal{S}$ un ensemble fini de symboles propositionnels, Σ un ensemble fini de clauses construites sur $\mathcal{S}$ contenant au plus r littéraux et k un entier naturel.
Question : Existe-t-il une interprétation de Σ qui satisfasse au moins k clauses de Σ.

C.H. Papadimitiou réduit **MAX**2**SAT** en *3***SAT** et prouve ainsi que **MAX**2**SAT** est **NP-complet** (Theorem 9.2 page 186 de [Papadimitriou 1994]). De manière plus générale, nous avons :

Propriété 4.8
Si $r \geqslant 2$ alors **MAX**r**SAT** $\in$ **NP-complet**.

Une fois de plus, il faut noter que bien que *2***SAT** appartienne à **P**, **MAX**2**SAT** appartient à **NP-complet**.

4.3 Quelques problèmes liés à SAT

Nous terminons ce chapitre par la présentation de quelques problèmes gravitant autour de SAT. Nous ne faisons pas ici une étude détaillée de ces problèmes mais nous nous contentons simplement de les définir. Cependant certains de ces problèmes feront l'objet d'études plus complètes dans d'autres chapitres de cette thèse.

4.3.1 Les champs de production

Les champs de production ont été introduits par P. Siegel [Siegel 1987], le problème correspondant se définit de la façon suivante :

Définition 4.9 (Champs de production)
Instance : Un ensemble $\mathcal{S}$ de symboles propositionnels, Σ une formule de la logique propositionnelle construite sur $\mathcal{S}$, Ω un ensemble de clauses construites sur $\mathcal{S}$.
Question : Parmi les clauses de Ω, quelles sont celles qui sont conséquences logiques de Σ ?

4.3.2 Les ATMS

L'ATMS – « *Assumption-Based Truth Maintenance System* » – introduit par J. De Kleer [De Kleer 1986] est un outil permettant des raisonnements révisables sur des bases de connaissances. Sa particularité principale est d'effectuer une distinction entre des données hypothèses et des données non hypothèses. L'ATMS est essentiellement un outil de raisonnement abductif. Sa principale fonction consiste à déterminer les

conjonctions d'hypothèses qui, avec la connaissance, expliquent (dans le sens de déduire) une donnée quelconque.

Définition 4.10 (ATMS)
Instance : Un ensemble $\mathcal{S}$ de symboles propositionnels, Σ une formule de la logique propositionnelle construite sur $\mathcal{S}$, une partition de $\mathcal{S}$ en données hypothèses $\mathcal{H}$ et données non hypothèses $\overline{\mathcal{H}}$.
Question (« nogood ») : Déterminer tous les monômes m construits avec des symboles de $\mathcal{H}$, tels que $\Sigma \wedge m$ soit inconsistante et que pour tout sous-monôme sm constructible à partir de m, si $sm \neq m$ alors $\Sigma \wedge sm$ est consistante.
Question (« label ») : Soit p un symbole propositionnel, déterminer tous les monômes m construits avec des symboles de $\mathcal{H}$, tels que $(\Sigma \wedge m) \models p$ et que pour tout sous-monôme sm constructible à partir de m, si $sm \neq m$ alors $(\Sigma \wedge sm) \not\models p$.

Il faut remarquer qu'il existe des relations entre les ATMS et les champs de productions. En effet, une grande partie des informations devant être fournies par un ATMS peuvent être définies en termes de champs de production ou en termes de calcul d'impliqués premiers.

4.3.3 Recherche de modèles préférés

Dans certaines formalisations du raisonnement en intelligence artificielle (*e.g.* non-monotonie, etc.), il n'est pas rare de devoir différencier les modèles d'une formule. Il est alors nécessaire de disposer de statuts pour les modèles et il est alors posssible de parler de préférence entre modèles. Dans ce cadre, l'information qui nous intéresse ne sera plus l'obtention d'un modèle (ou de tous les modèles) mais l'obtention d'un modèle préféré (ou de tous les modèles préférés) d'une formule.

Définition 4.11 (Modèles préférés)
Instance : Un ensemble $\mathcal{S}$ de symboles propositionnels, Σ une formule de la logique propositionnelle construite sur $\mathcal{S}$ supposée consistante, $\prec$ une relation d'ordre (partiel ou total ou parfois un pré-ordre partiel ou total) sur les interprétations.
Question 1 : Trouver un modèle M de Σ tel que pour tout modèle M' de Σ, $M \prec M'$.
Question 2 : Trouver tous les modèles M de Σ tel que pour tout modèle M' de Σ, $M \prec M'$.

Une perspective privilégiée au travail décrit dans cette thèse est l'extension des travaux réalisés pour **SAT** à un cadre de logique propositionnelle non monotone. Plus précisément, un algorithme simple et souvent efficace pour rechercher des modèles préférés dans un cadre de logique propositionnelle non monotone étendue par adjonction de conditions d'anormalités a été mis au point (cf. [Mazure *et al.* 1997a, Mazure *et al.* 1997d, Grégoire & Mazure 1998]).

4.3.4 Recherche d'impliqués et d'impliquants

La recherche de l'ensemble des impliqués ou des impliquants d'une formule de la logique propositionnelle est un problème rencontré dans de nombreuses applications. Ce problème se formalise de la manière suivante :

Définition 4.12 (Calcul d'impliqués ou d'impliquants)
Instance : Un ensemble de symboles propositionnels $\mathcal{S}$, une formule Σ construite sur $\mathcal{S}$.
Question 1 : Déterminer les clauses c telles que $\Sigma \models c$.
Question 2 : Déterminer les monômes m tels que $m \models \Sigma$.

Les réponses à ces questions peuvent parfois aboutir à des clauses ou à des monômes dont l'information n'est pas toujours pertinente (*e.g.* une clause tautologique). Une reformulation de la question en terme de recherche d'impliquants et d'impliqués premiers permet de résoudre ce problème.

4.3.5 La conséquence logique ou interrogation d'une base de connaissances

Définition 4.13 (Conséquence logique)
Instance : Un ensemble de symboles propositionnels $\mathcal{S}$, Σ et Ω deux formules de la logique propositionnelle construites sur $\mathcal{S}$.
Question : Est-ce que Ω est une conséquence logique de Σ : $\Sigma \models \Omega$?

Nous avons vu dans les chapitres précédents que la question « $\Sigma \models \Omega$? » était équivalente à « $\Sigma \wedge \neg\Omega$ est-elle inconsistante ? ». Cependant le problème de la déduction logique ne se borne pas à un problème de consistance. Il arrive fréquemment que la base de connaissance évolue dans le temps par ajouts ou retraits d'informations, ou encore que Σ soit questionnée de multiples fois avec des requêtes différentes. En fait, ce problème de déduction n'est pas uniquement réduit à un problème de consistance. Il s'inscrit aussi dans une problématique plus générale de représentation des connaissances. En effet, le problème de la déduction logique semble plus proche du problème où l'objectif est de déterminer une représentation de la connaissance par laquelle la déduction peut s'effectuer en temps polynomial. La compilation logique de bases de connaissances est une des approches les plus employées pour résoudre ce problème, elle fait l'objet d'une étude dans la troisième partie de cette thèse.

Deuxième partie

SAT : Aspects Algorithmiques

Chapitre 5

État de l'art

Cette partie du manuscrit a pour objectif de présenter les principales méthodes de résolution de **SAT**. Il est possible de dégager deux types d'approches pour tester la satisfaisabilité d'une formule CNF : les méthodes dites logiquement complètes par opposition aux méthodes dites incomplètes, ces dernières ne pouvant aboutir (sans certitude) que si la formule admet un modèle. Par ailleurs, nous montrons que ces deux manières d'aborder la résolution de **SAT** ne sont pas forcément antinomiques. En effet, il s'avère que l'une des voies les plus prometteuses en ce qui concerne la résolution de **SAT** [Freuder *et al.* 1995, Mazure *et al.* 1996a, Mazure *et al.* 1996b, Mazure *et al.* 1996c, Selman *et al.* 1997, Mazure *et al.* 1998a] se trouve dans la mise au point d'une méthode « mixte », c'est-à-dire d'une méthode permettant d'allier le caractère systématique des méthodes complètes et l'efficacité des méthodes incomplètes. Nous présenterons dans le chapitre 8 plusieurs combinaisons originales d'algorithmes complets et incomplets, dont certaines donnent des résultats pratiques extrêmement compétitifs.

Préalablement à la présentation des algorithmes, heuristiques ou raffinements des techniques existantes réalisés au cours de cette thèse, cette partie introductive aux méthodes de résolution de **SAT** nous permet d'établir un bref état de l'art sur ce problème. En effet, malgré la complexité théorique de ce problème, des progrès algorithmiques significatifs ont été obtenus ces dernières années et on assiste à un regain d'intérêt croissant quant à la compréhension du problème **SAT** . Pour ce dernier point, citons par exemple, le résultat obtenu par Chvátal et Szemerédi [Chvátal & Szemerédi 1988], concernant les instances *k***SAT** générées aléatoirement au pic de difficulté (cf. définition 3.13 et paragraphe 5.2.1). Ils ont montré que ces instances sont difficiles pour la résolution : la longueur des preuves croît, en moyenne, exponentiellement avec le nombre de variables. Il faut *prendre toute la mesure* de ce résultat car il s'applique non seulement aux méthodes utilisant le principe de résolution de Robinson [Robinson 1965] (cf. paragraphe 5.3.2) mais aussi à toutes les méthodes énumératives du type procédure de Davis & Putnam [Davis & Putnam 1960, Davis *et al.* 1962] (cf. chapitre 7).

De nombreux résultats ont été obtenus ces dernières années autour de la résolution du problème **SAT**. Au vue de l'étendue des travaux réalisés sur le sujet, il serait difficile d'en dresser une liste exhaustive. Nous limiterons notre présentation aux trois points suivants :

1. symétries ;
2. génération aléatoire et phénomènes de seuil ;
3. algorithmes.

5.1 Les symétries

Une large classe de problèmes présente une structure très symétrique : les formules qui les représentent restent invariantes sous certaines permutations de noms de variables. Plus formellement, une symétrie se définit de la manière suivante :

Définition 5.1 (Ensemble de littéraux complets)
Un ensemble P de littéraux *est dit* **complet** *si chaque littéral de P possède son complémentaire dans P :* $\forall l \in P,\ \neg l \in P$.

Définition 5.2 (Symétrie)
Soient P un ensemble complet de littéraux, S un ensemble de clauses construites à partir de P et σ une permutation définie sur P ($\sigma : P \longmapsto P$). σ est une **symétrie** *de S si elle satisfait les conditions suivantes :*

1. $\forall l \in P,\ \sigma(\neg l) = \neg\sigma(l)$;
2. $\sigma(S) = S$.

Krishnamurthy est à l'origine de cette définition des symétries [Krishnamurthy 1982], par la suite B. Benhamou et L. Saïs ont étendu ce principe et ont notamment montré comment il était possible de détecter et d'exploiter efficacement les symétries au sein de différents algorithmes comme la SL-résolution, la procédure de Davis & Putnam [Saïs 1993, Benhamou & Saïs 1994], ou la méthode SCORE(FD/B) (cf. paragraphe 8.1.1.3) [Chabrier *et al.* 1996, Chabrier 1997].

Un exemple classique de problème symétrique est celui des tiroirs et chaussettes (*pigeons-holes problem*), où l'objectif est de ranger N chaussettes dans P tiroirs ou encore de placer N pigeons dans P pigeonniers. Ce problème pour $N = 3$ et $P = 2$ se formalise de la façon suivante :

Exemple 5.1 (Pigeons 3–2 et symétries)
Si l'on considère que le littéral $p_{i,j}$ signifie *pigeon i dans pigeonnier j*, une représentation clausale S du problème des pigeons 3–2 est :

$$
\begin{array}{ll}
c_1 : & (p_{1,1} \vee p_{1,2}) \\
c_2 : & (p_{2,1} \vee p_{2,2}) \\
c_3 : & (p_{3,1} \vee p_{3,2}) \\
c_4 : & (\neg p_{1,1} \vee \neg p_{2,1}) \\
c_5 : & (\neg p_{1,1} \vee \neg p_{3,1}) \\
c_6 : & (\neg p_{2,1} \vee \neg p_{3,1}) \\
c_7 : & (\neg p_{1,2} \vee \neg p_{2,2}) \\
c_8 : & (\neg p_{1,2} \vee \neg p_{3,2}) \\
c_9 : & (\neg p_{2,2} \vee \neg p_{3,2})
\end{array}
$$

Soit σ une permutation définie comme suit :

$$
\begin{array}{l|l}
\sigma(p_{1,1}) = p_{2,1} & \sigma(\neg p_{1,1}) = \neg p_{2,1} \\
\sigma(p_{2,1}) = p_{3,1} & \sigma(\neg p_{2,1}) = \neg p_{3,1} \\
\sigma(p_{3,1}) = p_{1,1} & \sigma(\neg p_{3,1}) = \neg p_{1,1} \\
\sigma(p_{1,2}) = p_{2,2} & \sigma(\neg p_{1,2}) = \neg p_{2,2} \\
\sigma(p_{2,2}) = p_{3,2} & \sigma(\neg p_{2,2}) = \neg p_{3,2} \\
\sigma(p_{3,2}) = p_{1,2} & \sigma(\neg p_{3,2}) = \neg p_{1,2}
\end{array}
$$

La permutation ainsi définie est une symétrie de S. En effet :

$$\left.\begin{array}{l}\sigma(c_1) = c_2\\ \sigma(c_2) = c_3\\ \sigma(c_3) = c_1\\ \sigma(c_4) = c_6\\ \sigma(c_5) = c_4\\ \sigma(c_6) = c_5\\ \sigma(c_7) = c_9\\ \sigma(c_8) = c_7\\ \sigma(c_9) = c_8\end{array}\right\} \Longleftrightarrow \sigma(S) = S.$$

□

La plupart des algorithmes de démonstration automatique n'exploitent pas ces symétries, dont l'utilisation permettrait une amélioration des performances et la résolution de nombreux problèmes réputés difficiles.

En effet, la présence de symétries dans un ensemble de formules partitionne l'ensemble des solutions potentielles. Il est possible alors de n'explorer qu'une solution par classe, le résultat de la recherche pouvant être transposé à toute la classe via la symétrie. En pratique cela fournit donc un nouveau moyen d'élagage de l'espace de recherche, ce qui peut conduire à une réduction de complexité significative pour certaines classes de problèmes [Benhamou & Saïs 1992, Chabrier *et al.* 1996].

Des méthodes exploitant les symétries ont été mises en œuvre dans différents formalismes :

- les formules propositionnelles clausales [Benhamou & Saïs 1994] ;
- les formules propositionnelles « générales » [Benhamou *et al.* 1994, Belleannée & Vorc'h 1994] ;
- les réseaux de contraintes [Benhamou 1994] ;
- le langage SCORE(FD/B) [Chabrier 1997].

Dans tous les cas, la prise en compte des symétries par un démonstrateur réclame un algorithme de détection de symétries d'une part et une technique d'exploitation des symétries trouvées d'autre part.

La détection de symétries est un problème difficile montré équivalent polynomialement au problème d'isomorphisme de graphes [1] (classe de complexité ISO-complet) [Boy de la Tour & Demri 1995, Saïs 1993, Crawford 1992]. De ce fait, afin de ne pas nuire aux performances du démonstrateur dans le cas général les méthodes de détection proposées sont de nature incomplète : l'algorithme fournit les symétries qui lui sont possibles de déterminer en un temps limité a priori et échoue dans les autres cas.

L'expérimentation a montré que la prise en compte des symétries par un démonstrateur permet d'améliorer de façon spectaculaire les performances sur certaines classes de formules à fortes régularités, problèmes qui étaient jusqu'alors réputés comme extrêmement difficiles.

Une voie complémentaire pour améliorer les performances des démonstrateurs pour ces classes de formules consiste à étendre le langage de formules, de façon à pouvoir énoncer explicitement certaines symétries. Il est notamment possible d'ajouter au langage de nouveaux connecteurs exprimant directement des relations telles que « *au moins p propositions vraies parmi n* ». L'ajout de ces connecteurs, appelés

1. Ce résultat est intéressant, car le classement du problème de détection d'isomorphisme de graphes demeure une question ouverte (connu comme appartenant à NP et supposé non appartenir à P et à NP-complet). Certains auteurs pensent que ce problème appartient à une classe intermédiaire qui lie la classe P avec la classe NP.

opérateurs de cardinalité dans le cas clausal et opérateurs homogènes autrement, permet de généraliser la représentation clausale classique et d'établir des liens avec la programmation linéaire 0/1. Il faut noter que la modélisation d'un problème est primordiale, l'efficacité de la résolution en dépend. Par exemple le seul fait d'ajouter ces opérateurs de cardinalité a permis d'obtenir une preuve d'insatisfaisabilité polynomiale pour le problème des pigeons. Cependant, des résultats récents ont permis de montrer que le problème de la satisfaisabilité d'une formule constituée d'un ensemble de clauses de Horn et d'un opérateur de cardinalité appartient à la classe de complexité **NP-complet** [Bailleux & Marquis 1999].

5.2 Génération aléatoire et phénomènes de seuils

5.2.1 Les instances *k*SAT aléatoires

Une instance ***k*SAT aléatoire** est caractérisée par trois paramètres :

1. son nombre de variables V ;
2. son nombre de clauses C ;
3. et la taille des clauses k.

Le générateur standard d'instances ***k*SAT** consiste à générer des problèmes où toutes les clauses sont de taille fixe k. La génération d'une clause se fait en tirant, de manière aléatoire, k variables distinctes parmi l'ensemble des variables et en les négativant avec une probabilité de un demi.

Le modèle de génération aléatoire ***k*SAT** fait actuellement l'objet de nombreuses études, du fait qu'il permet de générer des problèmes très difficiles pour tous les algorithmes existants.

5.2.2 Les seuils au rapport C/V

Le phénomène de seuil est un phénomène remarquable qui a été découvert à l'origine pour certaines propriétés de graphes aléatoires. Celui de la connexité en est un exemple bien connu.

De façon générale un phénomène de seuil pour un propriété $Prop$ s'appliquant à des structures combinatoires aléatoires, signifie que lorsque certains paramètres évoluent, la probabilité que $Prop$ soit satisfaite passe brusquement de presque 1 à presque 0 (ou inversement). De plus, le saut de probabilité de 1 à 0 se produit sur un intervalle de variation de paramètres qui tend à devenir nul, lorsque la taille des structures augmente. Le seuil est défini par le lieu limite des paramètres où se produit le saut de probabilité pour une taille infiniment grande des structures. Les phénomènes de seuil induisent asymptotiquement ce qui est appelé des lois de probabilité $0-1$ [Lacoste 1996].

Un phénomène de seuil pour la cohérence des formules ***k*SAT** aléatoires a été mis en évidence indépendamment par de nombreux chercheurs (cf. [Dubois & Carlier 1991], [Mitchell *et al.* 1992], [André 1993], [Larrabee & Tusji 1993] et [Crawford & Auton 1993]). C'est le premier problème n'appartenant pas à la théorie des graphes, pour lequel on détecte une loi $0-1$ ne se déduisant pas d'une loi $0-1$ connue en logique !

On constate pour ces instances ***k*SAT** aléatoires, ayant un nombre de variables donné V, une brusque décroissance de la proportion d'instances satisfaisables en fonction du rapport du nombre de clauses C, que

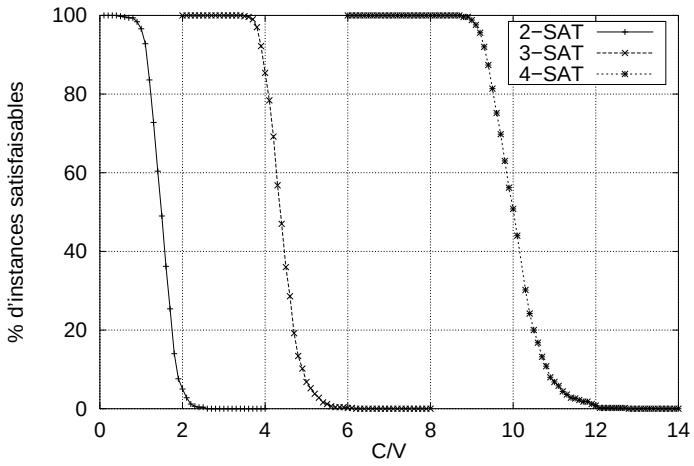

FIG. 5.1 – *Phénomènes de seuil pour k***SAT** *(*$k \in \{2, 3, 4\}$*)*

l'on fait croître, sur le nombre de variables V (cf. FIG. 5.1).

Lorsque l'on fait croître V, pour un k fixé, la transition de la zone où toutes les instances sont satisfaisables à celles où toutes sont non satisfaisables se produit dans le même domaine de valeurs du seuil C/V, mais la transition est de plus en plus brusque. C'est ce qui indique l'existence d'un phénomène de seuil comme le montre la figure 5.2 pour les instances *3***SAT**.

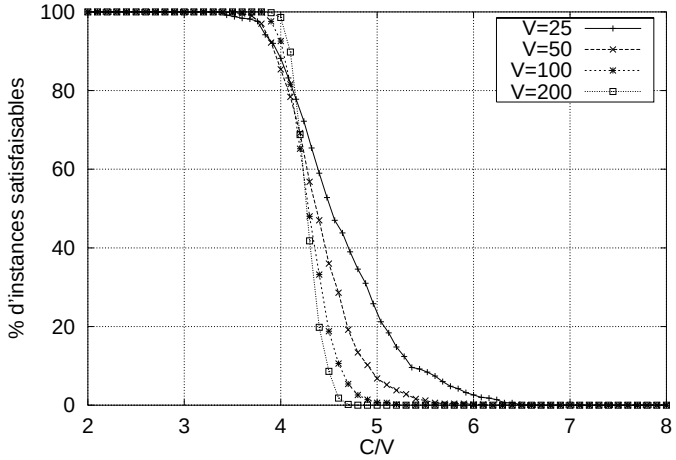

FIG. 5.2 – *Accroissement du phénomène de seuil en fonction du nombre de variables pour 3***SAT**

Le seuil est défini par la valeur du rapport C/V où les instances générées passent brutalement de la consistance vers l'inconsistance ; pour *3***SAT** ce seuil est proche de $4,25$[2] (valeur déterminée expérimentalement) [Cheeseman *et al.* 1991, Crawford & Auton 1993, Dubois *et al.* 1996, Mitchell *et al.* 1992, Dubois & Boufkhad 1996, AI 1996]. De même pour *4***SAT** , le seuil, exprimé par le ratio C/V avoisine expérimentalement la valeur de $9,80$ [Crawford & Auton 1993, Dubois *et al.* 1996]. De nombreuses personnes étudient ce phénomène de seuil et tentent de localiser théoriquement la valeur de ce ratio critique. La valeur pour *2***SAT** a été démontrée égale à 1 [Chvátal & Reed 1992], mais pour *k***SAT** $k \geqslant 3$ seules des bornes inférieures et supérieures ont pu jusqu'alors être démontrées. Dubois *et al.* [Dubois *et al.* 1996] ont établi une équation fonction de *k* qui approche les valeurs de ce ratio critique. Pour $k = 3$, la meilleure

2. Il est courant de considérer que le seuil pour des instances *3***SAT** de tailles inférieures à 500 variables avoisine $4,3$ tandis que pour des instances de plus grandes tailles le seuil de $4,25$ est jugé meilleur.

borne inférieure démontrée, à notre connaissance, est celle établie par Frieze et Suen qui s'élève à $3,003$ [Frieze & Suen 1996]. En ce qui concerne la borne supérieure de nombreux travaux ont été réalisés. Récemment O. Dubois et Y. Boufkhad [Dubois & Boufkhad 1997] ont établi une borne supérieure atteignant $4,64$. Cette valeur a été obtenue à partir d'une démonstration utilisant les *negatively prime solutions* (NPS). Reprenant cette idée de NPS, Kirousis *et al.* ont amélioré ce résultat pour atteindre la valeur de $4,60$. Cette valeur est à notre connaissance la meilleure borne supérieure jamais obtenue pour le seuil des instances *3***SAT** [Kirousis *et al.* 1998]. Il reste que ces valeurs sont encore loin du seuil constaté expérimentalement !

Par ailleurs, V. Chvàtal et E. Szemerédi ont démontré que pour tout $k \geqslant 3$ et pour tout ratio C/V supérieur ou égal $0,7 \times 2^k$, il existe un $\epsilon > 0$ tel qu'avec une probabilité qui tend vers 1 quand le nombre de variables tend vers l'infini, les instances générées par ce modèle aléatoire standard sont inconsistantes et que les preuves par résolution de la clause vide comportent au moins $(1+\epsilon)^V$ résolutions [Chvátal & Szemerédi 1988]. Ce qui prouve que ces instances sont très majoritairement difficiles pour la résolution. D'autre part, si l'on observe la courbe de difficulté de résolution de ces problèmes, on constate que celle-ci n'est pas uniforme (cf. FIG. 5.3) mais qu'au contraire le pic de difficulté est atteint au seuil, c'est-à-dire là où la probabilité de générer une instance consistante est de $1/2$ (cf. FIG. 5.4 page suivante). Toutefois la preuve que les instances les plus difficiles sont véritablement localisées au seuil reste à établir.

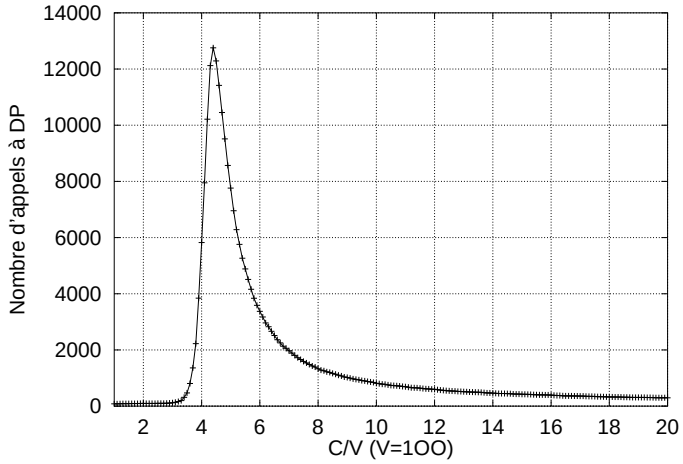

FIG. 5.3 – *Complexité des instances 3***SAT** *aléatoires*[3]

Ce phénomène de seuil fait l'objet de nombreuses recherches et présente de nombreux intérêts, par exemple :

- il fournit la possibilité de prédire la satisfaisabilité d'une instance *k***SAT** aléatoire en fonction du rapport C/V ;
- il permet de caractériser, pour la première fois des instances **SAT** difficiles à résoudre en moyenne par toutes les méthodes développées : instances localisées au seuil, c'est-à-dire au point de 50% de satisfaisabilité ;
- il a permis et permet la mise au point de méthodes de résolution du problème **SAT** de plus en plus efficaces.

3. La difficulté de l'instance est ici exprimée en nombre d'appels à la procédure de Davis & Putnam [Davis & Putnam 1960, Davis *et al.* 1962] (cf. paragraphe 7) avec l'heuristique de branchement dite « Jeroslow & Wang » [Jeroslow & Wang 1990].

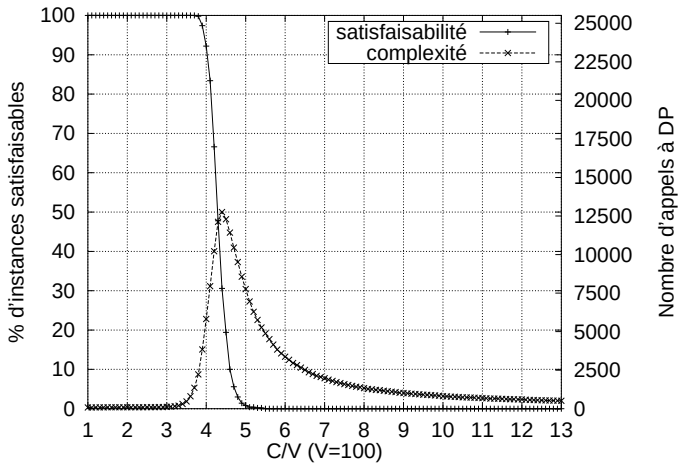

FIG. 5.4 – *Seuil et pic de difficulté pour des instances 3***SAT** *aléatoires*

5.2.3 Des modèles de génération aléatoire « non standard »

Il est clair que le modèle de génération d'instances aléatoires *k***SAT** standard fournit des instances difficiles pour les solveurs actuels. Cependant il est intéressant de savoir dans quelle mesure il est possible de décliner ce modèle de génération afin d'obtenir des instances au moins aussi difficiles ou des instances où il est possible d'exhiber également un phénomène de seuil ou encore des instances possédant certaines propriétés comme par exemple la garantie qu'elles admettent au moins une solution.

5.2.3.1 « Mixed clauses length model »

L'une des premières directions fut celle suivie par Gent & Walsh [Gent & Walsh 1994b] qui proposèrent un modèle de génération d'instances aléatoires où la taille des clauses n'est plus fixe : « *mixed clauses length model*[4] ». La taille des clauses est ici tirée aléatoirement avec une certaine probabilité. Différents modèles de génération peuvent être obtenus de cette manière, nous en expliciterons deux à titre d'exemples.

Le modèle *2-3-***SAT** consiste à générer avec une probabilité de $1/2$ une clause binaire ou ternaire. Une fois la longueur de la clause déterminée, la génération de cette clause se fait de la même manière que pour le générateur standard. Ce modèle de génération est une généralisation immédiate du générateur standard *k***SAT**, il n'est donc pas surprenant qu'un phénomène de seuil ait été également observé pour ce générateur (cf. FIG. 5.5).

De même, le pic de difficulté pour ces instances est atteint au rapport C/V correspondant à 50% d'instances satisfaisables (cf. FIG. 5.6), comme pour les instances du générateur standard. Curieusement ce pic de difficulté n'est pas toujours atteint à ce même endroit. En effet si on considère les instances *2-4-4-***SAT** , où une clause de taille 2 est générée avec une probabilité de un tiers et une clause de taille 4 est générée avec une probabilité de deux tiers, ce pic de difficulté n'est plus atteint là où 50% des instances sont satisfaisables mais aux alentours de $93,60\%$ d'instances satisfaisables (cf. FIG. 5.7 page 51) ! Une caractéristique similaire fut également observée par d'autres chercheurs qui exhibèrent des instances du modèle de génération standard extrêmement difficiles[5] et très loin du seuil [Gent & Walsh 1994a, Smith & Grant 1995].

4. Par opposition au modèle standard appelé «*fixed clauses length model* ».
5. C'est-à-dire plus difficile en moyenne que les instances au seuil.

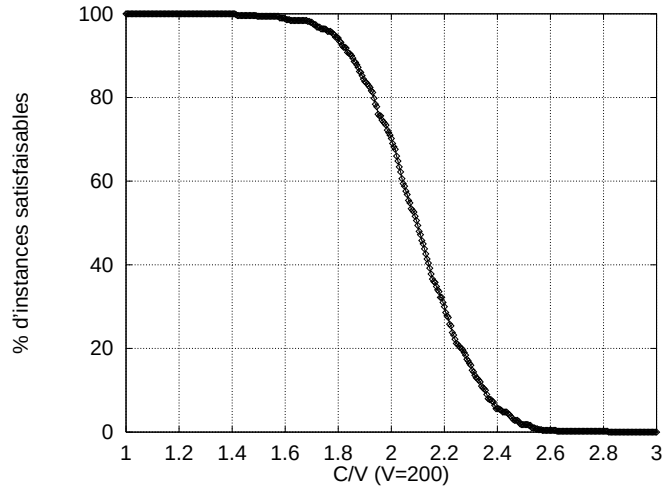

FIG. 5.5 – *Phénomène de seuil pour les instances du modèle de génération 2-3-***SAT**

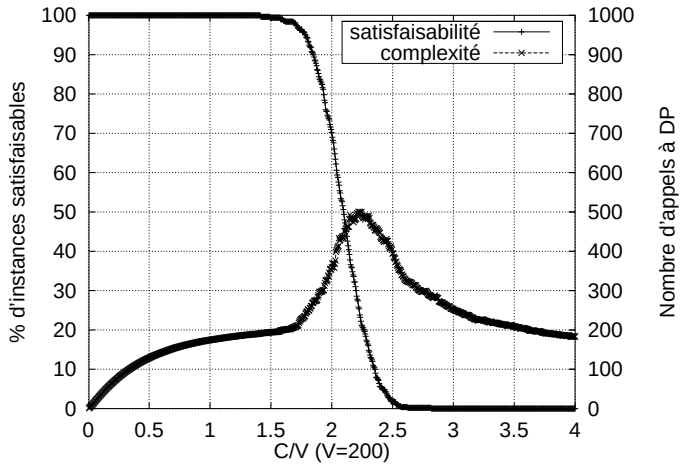

FIG. 5.6 – *Seuil et pic de difficulté pour des instances 2-3-***SAT**

Ce modèle de génération permet de générer des instances plus « proches » des problèmes réels où les clauses sont de tailles variables.

5.2.3.2 Modèle aléatoire « régulier »

Bien que le modèle aléatoire standard génère des instances relativement *régulières*, il subsiste tout de même une faible irrégularité en ce qui concerne l'apparition des littéraux. En effet, certains littéraux apparaissent plus souvent que d'autres. De ce fait une méthode comme la procédure de Davis & Putnam associée à une heuristique syntaxique exploite ce genre d'irrégularités. Afin de rendre plus difficiles encore les instances du modèle standard, R. Génisson et L. Saïs proposent d'imposer que tous les littéraux apparaissent le même nombre de fois dans les instances générées [Génisson & Saïs 1994]. Cette manière de procéder a effectivement permis d'obtenir des instances sensiblement plus difficiles à résoudre en pratique. Par ailleurs, un phénomène de seuil a pu également être mis en avant pour ce modèle de génération.

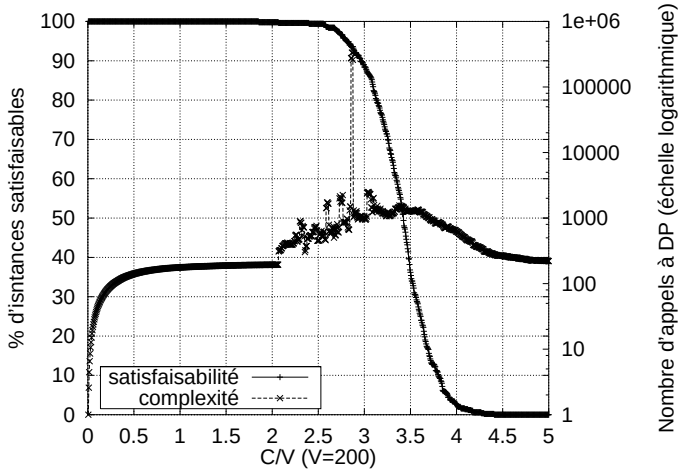

FIG. 5.7 – *Seuil et pic de difficulté pour des instances 2-4-4-***SAT**

5.2.3.3 Génération d'instances aléatoires consistantes

La génération d'instances de **SAT** aléatoires consistantes peut paraître sans intérêt, puisque nous savons à l'avance résoudre **SAT** étant donné que nous connaissons à l'avance la réponse à la question : « cette instance est-elle consistante ? ». Cependant, en raison du développement intensif des méthodes incomplètes de recherche locale pour **SAT**, il est dévenu nécessaire d'être capable de générer des instances consistantes « difficiles » afin d'évaluer ces approches. La solution la plus simple pour générer uniquement des instances consistantes est de supprimer dans une base de clauses toutes les clauses positives. C'est l'approche que propose Morris [Morris 1993]. En effet, un modèle trivial est l'interprétation qui falsifie toutes les variables. Cependant ce modèle fournit des instances faciles à résoudre. Ceci est dû à une trop grande disparité entre le nombre d'occurrences des littéraux positifs et négatifs de l'instance, la suppression de l'ensemble des clauses positives étant bien évidemment la cause de cette disparité. Par ailleurs, ces instances souffrent d'un autre léger défaut. Un solveur, aussi peu performant soit-il, peut répondre immédiatement : « Oui, cette instance admet un modèle », juste en détectant qu'il n'existe pas de clauses positives dans cette instance. Pour éliminer ce défaut, il suffit de renommer[6] certaines variables. Ainsi des clauses positives apparaissent et rendent l'instance plus naturelle et plus opaque pour les solveurs.

Malgré ce renommage, le déséquilibre entre le nombre de littéraux positifs et négatifs demeure. Indépendamment, A. Rauzy [Rauzy 1995a] et T. Castell [Castell & Cayrol 1996a, Castell 1997] ont cherché à contrôler cette différence. Le premier se restreint à la génération d'instances *3*-**SAT** en introduisant quatre nouveaux paramètres α, β, γ et δ afin de fixer la probabilité de générer une clause composée respectivement de 0, 1, 2, ou 3 littéraux positifs. L'objectif étant de générer des instances consistantes tout respectant une juste proportion entre les littéraux positifs et négatifs, A. Rauzy propose que les paramètres satisfassent le système suivant :

$$\begin{cases} \delta = 0 & \textit{\% Aucune clause totalement positive ne doit être générée} \\ 0 \leqslant \alpha, \beta, \gamma \leqslant 1 & \\ \alpha + \beta + \gamma = 1 & \\ 3\alpha + \beta = \gamma & \textit{\% Respect de l'équité entre le nombre de littéraux positifs et négatifs} \end{cases}$$

6. Pour quelques variables, v et $\neg v$ sont échangées (cf. définition 3.17 page 24).

Dans [Castell & Cayrol 1996a] une approche similaire est proposée, introduisant un seul nouveau paramètre la probabilité $ppos$ de tirer un littéral positif. Cette approche est une version restrictive de la méthode de A. Rauzy. La génération d'instances s'effectue comme dans le modèle de génération aléatoire standard, aux exceptions faites que la probabilité de tirer un littéral positif n'est plus égale à un $1/2$ mais à $ppos$ et que les clauses totalement positives ne sont pas acceptées.

T. Castell et M. Cayrol ont cherché à faire correspondre les deux modèles de génération, en calculant les valeurs de α, β et γ en fonction de $ppos$. Ils aboutissent aux résultats suivants :

$$\begin{cases} \alpha = \frac{(1-ppos)^3}{(1-ppos^3)} \\ \beta = \frac{ppos(1-ppos)^2}{(1-ppos^3)} \\ \gamma = \frac{3ppos^2(1-ppos)}{(1-ppos^3)} \end{cases}$$

En réinjectant ces valeurs dans le système d'équations proposé par A. Rauzy, ils ont ainsi pu obtenir la valeur de $ppos$ permettant de respecter l'équité entre le nombre de littéraux positifs et négatifs dans les instances de son modèle de génération. Cette valeur n'est rien d'autre que l'inverse du fameux *nombre d'or* ! La liste déja longue des apparitions de ce nombre est à compléter avec la génération aléatoire d'instances de **SAT**.

Par ailleurs, T. Castell dans [Castell 1997] a mené une étude empirique afin d'évaluer la difficulté des instances produites par son modèle de génération en fonction de $ppos$ et du rapport C/V. Les conclusions de cette étude restent mitigées. En effet, il n'a pas été possible de dégager une valeur des paramètres ($ppos$ et C/V) telle qu'un large panel de stratégies éprouvent de la difficulté à résoudre les instances générées.

5.3 Algorithmes

De nombreuses études concernent les aspects algorithmiques de la résolution du problème **SAT**. Ces travaux ont conduit à la proposition de multiples méthodes et stratégies que l'on classe selon leur éventuelle complétude.

5.3.1 Algorithmes incomplets

Les algorithmes incomplets s'arrêtent après un temps fixé (souvent polynomial) et donnent deux types de réponse : « Oui, l'instance admet un modèle » ou « Impossible de conclure ». Dans ce dernier cas, l'instance admet peut être une solution mais l'algorithme a été incapable de la fournir dans le temps qui lui était imparti.

Parmi les algorithmes incomplets les plus efficaces pour **SAT**, on trouve essentiellement ceux à base de recherche locale (recuit simulé, algorithme génétique, tabou, etc.).

La recherche locale fait l'objet du prochain chapitre où sont présentés divers algorithmes ainsi que nos contributions dans ce cadre.

Par ailleurs, la recherche locale n'est pas la seule méthode incomplète pour **SAT**. En effet, une méthode énumérative comme la procédure Davis & Putnam peut être rendue incomplète en limitant, par exemple, la profondeur de l'arbre de recherche.

5.3.2 Algorithmes complets

A l'inverse des méthodes incomplètes, les algorithmes complets s'arrêtent après un temps pouvant être exponentiel par rapport à la taille de l'instance et donnent une réponse dans tous les cas, que l'instance soit satisfaisable ou non.

Parmi les algorithmes complets, nous distinguons deux types : les algorithmes syntaxiques (basés sur le principe de résolution de J.A. Robinson [Robinson 1963, Robinson 1965]) très utilisés en logique du premier ordre et les algorithmes sémantiques (à la Davis & Putnam). Ces derniers apparaîssent être les plus efficaces pour tester de manière logiquement complète la satisfaisabilité d'une formule.

5.3.2.1 Le principe de résolution

Le principe de résolution (anciennement appelé consensus) est un principe élémentaire mais fondamental en logique. C'est un simple processus itératif, à chaque étape une nouvelle clause est générée par l'application d'une des deux règles suivantes.

Soit Σ un ensemble de clauses :

- **Règle de résolution**[7] :
  si deux clauses c_1 et c_2 appartenant à Σ se résolvent en un littéral l, alors ajouter à Σ la résolvante en l de c_1 et c_2 ;
- **Règle de fusion** :
  si $c = \{\alpha_1, \alpha_2, \ldots, \alpha_n, l, \beta_1, \beta_2, \ldots, \beta_m, l, \gamma_1, \gamma_2, \ldots, \gamma_p\}$ est une clause de Σ alors remplacer c par $c_f = \{\alpha_1, \alpha_2, \ldots, \alpha_n, l, \beta_1, \beta_2, \ldots, \beta_m, \gamma_1, \gamma_2, \ldots, \gamma_p\}$

Le processus se termine soit par l'obtention d'une clause vide, auquel cas l'ensemble initial de clauses est insatisfaisable, soit qu'aucune clause nouvelle ne puisse plus être produite, auquel cas l'ensemble de clauses est satisfaisable. Lorsque plus aucune clause ne peut être produite, on dit que l'ensemble de clauses est saturé ou que l'on a effectué une saturation de celui-ci. Cette méthode est en fait basée sur le théorème suivant :

Théorème 5.1
Soient deux clauses c_1 et c_2, si c_1 et c_2 se résolvent, alors la résolvante de c_1 et c_2 est une conséquence logique de c_1 et c_2.

Par ailleurs, la chaîne de clauses permettant de produire la clause vide est appelée une réfutation de l'ensemble de clauses. Il est également possible d'ajouter une troisième étape au principe de résolution qui consiste à supprimer toutes les clauses redondantes de la base de connaissances. Les clauses redondantes sont celles qui peuvent être subsumées par d'autres clauses de la base.

Ce principe de résolution est relativement inefficace car le nombre de résolvantes à effectuer est souvent énorme ; l'espace mémoire requis devient donc très important et les temps de calculs prohibitifs. De nom-

7. Chaque résolvante produite permet d'expliciter une information issue des clauses qui l'ont produite.

breuses stratégies ont été proposées afin de minimiser le nombre de clauses produites, parmi elles citons :

- la N-résolution [Robinson 1983] ;
- la résolution régulière [Davis *et al.* 1962, Tseitin 1968, Galil 1977] ;
- la résolution sémantique (PI-clash) [Chang & Lee 1973] ;
- la résolution linéaire et la SL-résolution [Kowalski & Kuehner 1971, Cubbada & Mouseigne 1988] ;
- la résolution étendue [Tseitin 1968] ;
- la résolution par entrée [Chang & Lee 1973] ;
- la résolution unitaire [Dowling & Gallier 1984, Escalada-Imaz 1989] ;
- la résolution dirigée [Dechter & Rish 1994] ;
- la résolution bornée [Génisson & Siegel 1994] ;

5.3.2.2 La procédure de Davis & Putnam

La procédure de Davis & Putnam [Davis & Putnam 1960] est un algorithme énumératif simple qui consiste en une exploration systématique de l'ensemble des interprétations afin de déterminer si l'une d'entre elles est un modèle. Cette procédure peut aisément être assimilée à une forme de résolution que l'on nomme résolution dirigée [Dechter & Rish 1994].

Une grande partie des algorithmes complets les plus efficaces pour **SAT** s'appuient sur une version améliorée de Davis & Putnam [Davis *et al.* 1962]. Une étude plus détaillée des différentes variantes de cet algorithme est faite dans le chapitre 7. Nous présentons également notre contribution à l'amélioration de cette procédure. Par ailleurs, dans le chapitre 8, nous montrons comment faire coopérer des méthodes de recherche locale avec des algorithmes complets de type DP.

Chapitre 6

Recherche locale

La génération d'instances aléatoires et la mise en avant des phénomènes de seuil (cf. paragraphe 5.2 page 46) qui montrent une séparation tranchée des instances satisfaisables de celles insatisfaisables, ont fortement contribué à la mise au point de techniques spécialisées. Ces techniques se sont spécialisées soit dans la recherche d'une solution (techniques à base de recherche locale), soit dans la preuve de la contradiction (DP et ses variantes). La recherche locale constitue donc une alternative aux méthodes énumératives pour l'obtention d'un modèle.

Les techniques à base de recherche locale ont été développées initialement afin de résoudre des problèmes d'optimisation combinatoires. Dans le cas de **SAT**, les méthodes à base de recherche locale sont des techniques incomplètes se restreignant au traitement de la consistance, c'est-à-dire qu'elles permettent d'exhiber une solution mais ne garantissent pas son obtention, et ne peuvent a priori prouver l'inconsistance. Il faut noter que le principe de recherche locale dédié à la recherche de solutions n'est pas récent. Des méthodes comme le recuit-simulé, la recherche tabou et les algorithmes génétiques [Goldberg 1989] existent depuis fort longtemps et ont été appliquées avec succès dans d'autres domaines. Mais ce n'est que très récemment que l'efficacité de ces méthodes a été mise au service de **SAT** [Hansen & Jaumard 1990, Minton *et al.* 1990, Selman *et al.* 1992, Gu 1992, Jaumard *et al.* 1993].

La recherche locale pour **SAT**, s'appuie sur un parcours non systématique (souvent stochastique) de l'espace de recherche, c'est-à-dire de l'espace des interprétations. À chaque pas, seulement quelques interprétations sont examinées. En contrepartie, comparativement aux approches complètes, les algorithmes à base de recherche locale jouissent d'une plus grande flexibilité dans l'exploration de cet espace de recherche. En effet, ces méthodes de recherche n'imposent pas, a priori, d'ordre sur l'examen des interprétations, par opposition par exemple à la procédure de Davis & Putnam (cf. chapitre 7) où les interprétations sont testées suivant une ordre induit par l'heuristique de branchement. Comparativement aux recherches complètes, le nombre d'interprétations visitées par la recherche locale est très réduit. Malgré cela, ces algorithmes se sont révélés d'une efficacité redoutable pour la recherche de modèles.

Toutefois, il ne faut pas se leurrer quant à l'efficacité de la recherche locale. Même si aujourd'hui, elle est la seule méthode permettant dans la pratique de trouver un modèle pour certaines instances consistantes, l'espace de recherche à examiner peut être si vaste que ces méthodes, comme tous les autres, admettent des instances très difficiles à résoudre. Par ailleurs, si la recherche a échoué il est alors impossible de conclure sur la satisfaisabilité ou non de l'instance. Seules des conjectures peuvent être avancées, ce qui est très insuffisant dans le cadre du test de consistance.

Nous commençons ce chapitre par un bref état de l'art des méthodes de recherche locale pour **SAT** et plus précisément par la présentation de l'algorithme GSAT [Selman *et al.* 1992, Selman *et al.* 1993]. Cet algorithme est l'un des premiers à avoir permis la résolution de larges instances consistantes qui étaient jusqu'alors inaccessibles. De nombreuses variantes de cette méthode ont été proposées, nous en décrivons quelques-unes. La deuxième partie de ce chapitre présente un algorithme, appelé TSAT, basé sur GSAT et sur la recherche tabou. Cet algorithme que nous avons mis au point donne des résultats très satisfaisants sur un large panel d'instances [Mazure *et al.* 1995, Mazure *et al.* 1997e]. Avant de conclure ce chapitre dédié à la recherche locale pour **SAT**, nous présenterons des résultats que nous avons obtenus au sujet de la génération de l'interprétation initiale par des algorithmes de recherche locale.

6.1 État de l'art

La recherche locale pour **SAT** est une approche élémentaire qui consiste à se déplacer d'interprétation en interprétation jusqu'à l'obtention d'un modèle. Le paysage ainsi décrit est un sous-ensemble de l'ensemble des interprétations et les modèles sont les minima d'une fonction d'évaluation. L'objectif de ces méthodes est d'exhiber un modèle et donc d'atteindre un point dont l'altitude est minimal. Il est courant de parler d'altitude et de paysage « montagneux » lorsque l'on décrit le comportement d'une méthode de recherche locale. En effet s'il était possible de représenter l'espace complet des interprétations en fonction du nombre de clauses falsifiées, ce paysage ressemblerait certainement à un paysage alpin. Par conséquent, l'objectif de ces méthodes est d'atteindre le niveau de la mer, c'est-à-dire une altitude zéro signifiant qu'aucune clause n'est falsifiée et donc qu'un modèle a été trouvé.

La fonction d'évaluation est très souvent fonction du nombre de clauses falsifisées. Pour trouver un point d'altitude minimale, il faut donc minimiser cette fonction d'évaluation. Le premier principe est la « descente » c'est-à-dire que tant qu'il existe une interprétation au voisinage de l'interprétation courante améliorant la fonction d'évaluation, il faut se déplacer vers cette interprétation voisine. Ce déplacement est appelé réparation de l'interprétation courante.

Algorithme 6.1 RECHERCHE LOCALE

```
1.  Function RECHERCHE_LOCALE : boolean
2.  Input  : Σ un ensemble de clauses et max_reparations, le nombre maximum
             de réparations autorisées ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.    Générer une configuration initiale I ;      %% une interprétation complète de Σ
6.    nb_reparations=0 ;                          %% Nombre de réparations effectuées
7.    while (nb_reparations < max_reparations) and (I n'est pas un modèle)
8.    do
9.      if (une descente est possible)
10.     then Remplacer I par une interprétation voisine permettant la descente ;
11.     else Remplacer I par une interprétation suivant le critère d'échappement ;
12.     fi
13.     nb_reparations++ ;
14.   done
15.   return (I est modèle) ;  %% Retouner la valeur du test : « I est-il un modèle ? »
16. End
```

Le voisinage est à définir et varie suivant les algorithmes de recherche locale. Pour **SAT**, le voisinage considéré est le plus souvent l'ensemble des interprétations distantes de 1, en terme de la distance de HAMMING (cf. définition 2.7).

Appliqué seul, le principe de descente conduit souvent à un minimum local. Ce minimum local est une interprétation qui n'est pas un modèle et pour laquelle il n'existe pas dans son voisinage d'interprétation permettant de minimiser la fonction d'évaluation. Il faut alors appliquer un second principe pour sortir du minimum local : le principe d'échappement. Il doit permettre de s'éloigner suffisamment de l'interprétation courante afin d'éviter de la revisiter une nouvelle fois. Ce principe autorise donc des dégradations de la valeur de la fonction objective. Il existe différentes stratégies pour s'échapper d'un minimum local (aléatoire, retour arrière, tabou, etc.) et différentes manières de les utiliser (tout au long de la recherche, uniquement lorsqu'un minimum local a été atteint, etc.). C'est souvent ce principe qui différencie les multiples algorithmes de recherche locale pour **SAT**. L'algorithme 6.1 page précédente décrit succinctement le calcul effectué par une méthode de recherche locale.

Réaliser le meilleur déplacement (c'est-à-dire choisir le « meilleur » voisin) constitue la motivation principale des méthodes de recherche locale. En effet, déterminer l'interprétation qui parmi un ensemble d'interprétations est celle qui est la plus proche de l'objectif (c'est-à-dire celle qui permettra d'atteindre un modèle avec le plus petit nombre de déplacements), est une tâche délicate étant donné que le nombre d'interprétations pouvant être examinées est extrêmement réduit par rapport au nombre d'interprétations admises par la formule. Par ailleurs, la recherche aboutissant le plus souvent à un minimum local, les techniques d'échappement jouent un rôle primordial pour l'efficacité des méthodes. La littérature sur le sujet foisonne, nous présentons une liste non exhaustive de différents algorithmes.

6.1.1 Algorithme glouton : « Hill-Climbing »

Dans cette méthode, une configuration initiale est considérée et soumise à des changements locaux améliorant à chaque fois la fonction objective ou d'évaluation, jusqu'à trouver un optimum, le plus souvent local. Cet algorithme ne fait donc que l'étape de descente sans utiliser de mécanisme d'échappement.

Algorithme 6.2 HILL-CLIMBING

```
1.  Function Hill_Climbing : boolean
2.  Input  : Σ un ensemble de clauses ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     Générer une configuration initiale I ;     %% une interprétation complète de Σ
6.     while (une descente est possible) and (I n'est pas un modèle)
7.     do
8.        Remplacer I par une interprétation voisine permettant la descente ;
9.     done
10.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
11. End
```

Une telle méthode souffre d'un inconvénient : l'arrêt dès le premier optimum rencontré. Au vu de l'étendue de l'espace possible des interprétations ($2^{(\text{le nombre de propositions de l'instance})}$), il est clair que généralement l'optimum atteint est local. Comme la procédure ne prévoit pas de moyen d'échappement, cet algorithme est très inefficace.

6.1.2 Le Recuit Simulé

Dans cette méthode, à l'origine utilisée pour caractériser l'évolution d'un système thermo-dynamique (d'où son nom), une configuration initiale est générée, puis un changement local est considéré et son effet sur la fonction objective est calculé. Si le changement provoque une amélioration, il est alors effectué, sinon une probabilité d'acceptation est calculée. Cette probabilité évolue en fonction de la détérioration de la configuration et du temps de recherche (plus la configuration se détériore et le temps passé augmente, plus la probabilité décroît). Une fois cette probabilité calculée, un nombre aléatoire entre 0 et 1 est généré. Le changement est accepté si et seulement si ce nombre aléatoire est inférieur à la probabilité calculée. Cette procédure s'arrête lorsqu'aucune réparation locale n'a été effectuée pendant un nombre donné d'itérations.

Algorithme 6.3 RECUIT SIMULÉ

```
1.  Function RECUIT_SIMULÉ : boolean
2.  Input  : • Σ un ensemble de clauses,
             • p : la valeur initale de la probabilité d'accepter une nouvelle
                   interprétation lorsque l'interprétation courante n'autorise
                   aucune descente,
             • une fonction f de recalcul de cette probabilité,
             • max_stag, le nombre maximum de stagnations autorisées ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     Générer une configuration initiale I ;     %% une interprétation complète de Σ
6.     stag=0 ;                  %% Nombre de stagnations sur l'interprétation courante
7.     nb_reparation=0 ;                          %% Nombre de réparations effectuées
8.     proba=p ;     %% Probabilité d'acceptation, recalculé tout au long de la recherche
9.     while (stag < max_stag) and (I n'est pas un modèle)
10.    do
11.       if (une descente est possible)
12.       then
13.           Remplacer I par une interprétation voisine permettant la descente ;
14.           stag=0 ;
15.           nb_reparation++ ;
16.       else
17.           choisir I' une interprétation voisine de I ;
18.           proba=f(proba,nb_reparation,I,I',Σ) ;  %% la probabilité est recalculée
                 %% en fonction du temps passé et de la détérioration de la configuration
19.           a=un nombre aléatoire entre 0 et 1 ;
20.           if (a < proba)
21.           then
22.               I=I' ;
23.               nb_reparation++ ;
24.               stag=0 ;
25.           else
26.              stag++ ;
27.           fi
28.       fi
29.    done
30.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
31. End
```

Contrairement au Hill-Climbing, ici la procédure ne s'arrête pas au premier optimum rencontré, l'ensemble des solutions potentielles est donc mieux exploré. Cependant la possibilité de ne pas obtenir l'optimum global persiste. De fait, aucun algorithme de recherche locale ne peut garantir l'obtention de la solution.

L'inconvénient majeur de cette technique est qu'elle possède de nombreux paramètres qui sont extrêmement difficiles à régler. De plus, comme pour la plupart des méthodes de recherche locale, les valeurs des paramètres ont une grande influence sur la performance de la procédure.

Par ailleurs, l'algorithme proposé donne juste une représentation de l'ossature d'un algorithme à base de Recuit Simulé [Kirkpatrick *et al.* 1983]. En effet, il est possible de dériver de ce modèle de représentation de nombreuses variantes, aussi singulières les unes que les autres, dont le comportement en terme d'efficacité peut se révéler extrêmement fluctuant.

6.1.3 La méthode tabou

Le principe de cette méthode est relativement simple, il consiste en une descente jusqu'à atteindre un optimum. Si cet optimum n'est pas global, un changement local dans le sens de la plus petite régression s'effectue avec cependant un certain nombre de changements interdits (liste tabou). Cette liste d'interdits, gérée le plus souvent en FIFO[1], évolue tout au long de la recherche, ceci afin d'éviter le phénomène de « cuvette ». La procédure s'arrête quand il n'y a plus d'amélioration pendant un certain nombre d'itérations.

Algorithme 6.4 TABOU

```
1.  Function Tabou : boolean
2.  Input  : • Σ un ensemble de clauses,
             • l : la longueur de la liste tabou,
             • it_max : nombre d'itérations maximum sans amélioration ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     Générer une configuration initiale I ;     %% une interprétation complète de Σ
6.     nb_it=0 ;                                  %% Nombre d'itérations sans améliorations
7.     liste= new FIFO de taille l ;   %% Création de la liste tabou d'interprétations
8.     while (nb_it < it_max) and (I n'est pas un modèle)
9.     do
10.        mettre à jour liste avec I ;
11.        if (une descente est possible)
12.        then
13.           Remplacer I par une interprétation voisine permettant la descente ;
14.           nb_it=0 ;
15.        else
16.           Remplacer I par une interprétation voisine telle qu'elle engendre la
              plus petite régression de la fonction objectif et qu'elle ∉ liste ;
17.           nb_it++ ;
18.        fi
19.    done
20.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
21. End
```

1. *First In First Out.*

Comme pour le Recuit-Simulé, il existe de multiples variantes de l'algorithme tabou [Glover 1989, Glover 1990]. Nous décrivons plus en détails quelques grands principes de cette méthode (règles d'application, critère d'aspiration, etc.) lors du passage consacré à l'algorithme de recherche locale que nous avons mis au point (cf. paragraphe 6.2). Cet algorithme nommé TSAT (*Tabu Search for SAT*) est basé sur la méthode tabou, comme son nom l'indique.

6.1.4 GSAT et ses variantes

GSAT[2] est certainement la plus connue des méthodes de recherche locale pour **SAT**. Cet algorithme fut proposé initialement en 1992 par Selman *et al.* [Selman *et al.* 1992], mais c'est lors du « *Second Challenge on Satisfiability Testing* » de DIMACS en 1993 [DIM1993], que des variantes extrêmement efficaces de cet algorithme furent proposées [Selman *et al.* 1993]. En 1993, le papier traitant de cet algorithme a été couronné d'un prix à la conférence AAAI'93[3].

Il existe peu de différences entre GSAT et les méthodes de recherche locale « classique » présentées précédemment. La fonction d'évaluation de cette méthode reste la plus naturelle qu'il soit, c'est-à-dire le nombre de clauses falsifiées. Cette fonction d'évaluation est couramment dénommée « *min-conflict* » [Minton *et al.* 1990]. Le voisinage reste identique à la description que nous en avons faite préalablement. En effet le voisinage utilisé par GSAT est un « voisinage direct », c'est-à-dire que les interprétations voisines sont toutes les interprétations qui diffèrent de l'interprétation courante de 1 littéral. La fonction $Voisins$ se définit alors comme :

Définition 6.1 (Voisinage direct)
$Voisins_{direct}(I) = \{J \mid J \text{ est une interprétation et } d_H(I, J) = 1\}$.

Il est possible de généraliser cette notion de voisinage de manière à considérer toutes les interprétations distantes, en terme de HAMMING, d'une longueur inférieure ou égale à une constante k. Le problème de cette généralisation est que plus k est élevé, plus la sélection du meilleur voisin sera complexe et coûteuse en temps. De nombreuses expérimentations afin d'obtenir une valeur optimum de k ont été réalisées [Bailleux 1998]. Cependant, il n'existe pas à notre connaissance d'algorithmes de recherche locale considérant des voisinages pour $k > 1$.

La description de GSAT peut se faire de manière très succincte tant cet algorithme est simple. Cet algorithme commence par générer une interprétation complète des variables propositionnelles. Le choix de l'interprétation initiale s'opère de la façon suivante : pour chaque proposition, avec une probabilité de un demi, on choisit de l'affecter à $\mathbb{V}$ ou à $\mathbb{F}$. La plupart (voire même la totalité) des algorithmes de recherche locale utilisent ce principe pour générer la configuration initiale, nous verrons plus tard dans ce chapitre d'autres manières de choisir l'interprétation initiale (cf. paragraphe 6.3).

Une fois, la configuration initiale déterminée, l'algorithme effectue un certain nombre de réparations jusqu'à trouver un modèle. Une inversion ou une réparation d'une variable est simplement une inversion de la valeur d'une variable propositionnelle[4]. Le nombre d'inversions (`Max_Flips`) autorisées est fixé au départ de la procédure. Si l'algorithme ne trouve pas de solution, la procédure recommence avec une nouvelle interprétation initiale. Ce processus peut se répéter un nombre maximal de fois (`Max_Tries`) fixé

2. *Greedy algorithm for SAT*

3. Il est à noter que des algorithmes très similaires à GSAT ont été proposées antérieurement ou simultanément et indépendamment par d'autres chercheurs, citons notamment l'algorithme de Hansen & Jaumard pour **MAXSAT** [Hansen & Jaumard 1990] et l'algorithme de Gu pour **SAT** [Gu 1992].

4. Nous employons également le terme anglo-saxon « *flip* » et même le néologisme « *flipper* une variable » signifiant inverser la valeur de cette variable.

au démarrage de la procédure. L'algorithme 6.5 ci-dessous synthétise cette description.

Dans cet algorithme, nous observons (ligne 22) que le choix parmi les meilleurs voisins est laissé au hasard. I.P. Gent et T. Walsh proposent d'autres critères de choix plus déterministes [Gent & Walsh 1993b]. Ils suggèrent notammment de choisir la variable qui a été la plus anciennement flippée dans le temps. Dans l'éventualité où plusieurs variables n'ont jamais été flippées, un ordre arbitraire mais fixe détermine la variable à sélectionner.

Algorithme 6.5 GSAT

```
1.  Function GSAT : boolean
2.  Input  : Σ un ensemble de clauses, deux entiers : Max_Tries et Max_Flips ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     for i from 1 to Max_Tries
6.     do
7.         I=une interprétation complète générée aléatoirement ;
8.         for j from 1 to Max_Flips
9.         do
10.           if (I est un modèle)
11.           then
12.               return true ;
13.           else
14.              foreach variable v de Σ
15.              do
16.                 fals_to_sat[v]=le nombre de clauses falsifiées par I qui
                                   deviendraient satisfaites, si v était flippée ;
17.                 sat_to_fals[v]=le nombre clauses satisfaites par I qui
                                   deviendraient falsifiées, si v était flippée ;
18.                 score[v]=fals_to_sat[v]-sat_to_fals[v] ;        %% min-conflict
19.              done
20.              list_of_max_diff=liste des variables ayant le meilleur score ;
21.              x=une variable au hasard parmi list_of_max_diff ;
22.              I=(I avec la valeur d'affectation de x inversée) ;
23.           fi
24.        done
25.    done
26.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
27. End
```

GSAT ne présente pas, comme pour les méthodes précédemment décrites, deux phases bien distinctes : descente et échappement. En effet, la descente n'est pas ici une descente stricte, elle autorise des remontées de la fonction d'évaluation. Plus précisément, si le `score` est positif, le nombre de clauses falsifiées va effectivement diminuer (nous sommes dans une phase de descente). Si le `score` est nul, le nombre total de clauses insatisfaites reste constant (nous sommes dans une configuration appelée plateau et l'algorithme explore ce plateau afin de trouver un voisin de même altitude lui permettant d'échapper à ce minimum local [Minton *et al.* 1990]). Enfin, si le `score` est négatif, l'algorithme s'éloigne de son objectif, nous sommes donc en phase de remontée. Des `score`s négatifs sont relativement courants et souvent indispensables pour assurer l'efficacité de cette méthode.

GSAT autorise donc les dégradations de la fonction objective au cours de la recherche, ce qui peut être

considéré comme un moyen de s'échapper d'un optimum local.

Par ailleurs, il existe une phase d'échappement dans cette méthode. Elle consiste en un redémarrage de la procédure avec une nouvelle configuration initiale. Cet étape relativement brutale est un procédé radical pour sortir des optima locaux. De plus, un choix aléatoire est effectué parmi les variables présentant le meilleur `score`. Ce caractère aléatoire permet assurément d'éviter de nombreux flips récurrents.

Il reste que cette approche conduit le plus souvent à un optimum local. C'est pourquoi de nombreuses variantes de cet algorithme ont vu le jour [Selman & Kautz 1993b], nous ne décrivons ici que celles qui nous semblent les plus représentatives.

6.1.4.1 Random Walk Strategy

Nous présentons ici, une des meilleures améliorations de GSAT [Selman *et al.* 1994] : « *Random Walk Strategy* ». Lorsqu'il est fait référence à GSAT, on fait fréquemment référence implicitement à GSAT+*Random Walk Strategy* (GSAT+RWS). Afin d'éviter toute confusion, nous distinguons clairement les deux approches dans cette thèse.

Random Walk Strategy, comme son nom l'indique s'appuie sur une série de déplacements aléatoires pour s'extraire des extremums locaux. Ces déplacements aléatoires s'effectuent de la manière suivante :

- avec une probabilité p, on choisit aléatoirement de flipper une variable parmi l'ensemble des variables apparaissant dans les clauses falsifiées, c'est-à-dire les variables ayant une valeur de `fals_to_sat` (cf. algorithme 6.5) strictement positive ;
- avec une probabilité $1 - p$, on élit la variable à réparer de la même manière que dans GSAT.

L'adaptation de GSAT vers GSAT+RWS est présentée dans l'algorithme 6.6 de la présente page.

L'introduction de ces déplacements aléatoires dans GSAT a permis de montrer expérimentalement que l'algorithme converge plus rapidement et nécessite peu d'essais [Selman *et al.* 1994]. Pour un comportement expérimental optimal de l'algorithme, B. Selman *et al.* proposent d'utiliser une valeur de la probabilité p comprise entre $0,5$ et $0,6$.

B. Selman et H. Kautz dans [Selman & Kautz 1995], proposent diverses variantes autour des déplacements aléatoires. Une première variante consiste à n'utiliser les réparations aléatoires que lorsque l'algorithme a atteint un minimum local, c'est-à-dire lorsque ($\texttt{max\_diff} \leqslant 0$). Une seconde variante diffère par l'ensemble des variables susceptibles d'être réparées lors d'un déplacement aléatoire. Initialement, il est proposé de choisir une variable parmi celles qui, par leur inversion, vont satisfaire au moins une clause qui était falsifiée par l'interprétation courante. L'option « *all* » consiste à choisir une variable parmi l'ensemble complet des variables sans aucune distinction suivant leurs différents scores.

Cependant, il semble que l'option *Random Walk Strategy* la plus performante soit celle proposée initialement [Selman & Kautz 1995]. Soulignons par ailleurs que l'introduction de ce caractère aléatoire peut toutefois rendre GSAT moins efficace sur certaines instances [Cha & Iwama 1995].

Algorithme 6.6 GSAT+RWS

```
1.  Function GSAT+RWS : boolean
2.  Input  : Σ un ensemble de clauses, deux entiers : Max_Tries et Max_Flips,
             une probabilité p de réparer aléatoirement ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     for i from 1 to Max_Tries do
6.         I=une interprétation complète générée aléatoirement ;
7.         for j from 1 to Max_Flips do
8.            if (I est un modèle) then return true ;
9.            else
10.              foreach variable v de Σ do
11.                 fals_to_sat[v]=le nombre de clauses falsifiées par I qui
                                   deviendraient satisfaites, si v était flippée ;
12.                 sat_to_fals[v]=le nombre clauses satisfaites par I qui
                                   deviendraient falsifiées, si v était flippée ;
13.                 score[v]=fals_to_sat[v]-sat_to_fals[v] ;        %% min-conflict
14.             done
15.             list_of_max_diff=liste des variables ayant le meilleur score ;
16.             list_of_FalsToSat_pos=liste des variables t.q. fals_to_sat[v]>0 ;
17.             n=un nombre généré aléatoirement entre 0 et 1 ;
18.             if (p > n)
19.             then                                          %% Random Walk Strategy
20.                x=une variable au hasard parmi list_of_FalsToSat_pos ;
21.             else                                                %% GSAT classique
22.                x=une variable au hasard parmi list_of_max_diff ;
23.             fi
24.             I=(I avec la valeur d'affectation de x inversée) ;
25.          fi
26.        done
27.    done
28.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
29. End
```

6.1.4.2 Break Out Method

La méthode proposée par P. Morris, appelée « *Break Out Method* » [Morris 1993], est une stratégie différente de celle de GSAT. Cependant une approche similaire a été développée pour GSAT par B. Selman et H.A. Kautz[5] [Selman & Kautz 1993a].

Afin d'échapper aux minima locaux, P. Morris suggère une technique de pondération des clauses. Cette pondération permet de modifier dynamiquement la fonction d'évaluation durant la recherche.

Au départ de la procédure, chaque clause se voit créditer d'un poids de 1. Lorsqu'un extremum local est atteint, les clauses falsifiées par l'interprétation courante voient leur poids augmenté de 1. La variable à réparer est celle qui fournira une interprétation maximisant la somme des poids des clauses satisfaites et/ou minimisant la somme des poids des clauses falsifiées par cette nouvelle interprétation. Le voisinage utilisé dans cette méthode reste identique à celui de GSAT, à savoir un voisinage direct (cf. algorithme 6.7).

5. Selman et Kautz nomment leur approche « *weight* » dans leur implantation de GSAT [Selman & Kautz 1995].

Cette stratégie s'avère parfois plus performante que GSAT+RWS, particulièrement sur des instances à peu de solutions ou à solutions isomorphes [Cha & Iwama 1995].

Contrairement aux autres algorithmes de réparation locale (exception faite de Tabou), cet algorithme guide son exploration future par celle déjà effectuée. L'objectif de cette méthode est donc « d'apprendre » afin d'atténuer et même d'éliminer l'attraction des minima locaux rencontrés ultérieurement.

Il est à noter qu'une autre voie pour atteindre cet objectif est d'ajouter à la base de connaissances, des clauses permettant de « gommer » ce minimum local [Ginsberg & McAllester 1994, Cha & Iwama 1995, Hao & Tétard 1996]. Par ailleurs, l'ajout des clauses peut permettre de transformer la recherche locale en une méthode complète. Cependant ces méthodes à base de production de clauses sont extrêmement gourmandes en espace mémoire et leur efficacité reste à prouver.

Algorithme 6.7 BREAK OUT METHOD

```
1.  Function Break_Out_Method : boolean
2.  Input  : Σ un ensemble de clauses de poids 1 ;
3.  Output : true si un modèle de Σ est trouvé ;     %% la fonction boucle sinon ...
4.  Begin
5.      I=une interprétation complète générée aléatoirement ;
6.      while (I n'est pas un modèle)
7.      do
8.         if (Aucune descente n'est possible)
9.         then
10.           foreach clause c falsifiée
11.           do
12.               augmenter le poids de c ;
13.           done
14.        fi
15.        Remplacer I par l'interprétation minimisant (et/ou maximisant) la somme
           des poids des clauses falsifiées (satisfaites) ;
16.    done
17.    return true ;
18. End
```

De nombreuse déclinaisons de la procédure de P. Morris ont été proposées, citons par exemple celle de Selman et Kautz [Selman & Kautz 1993a] qui suggèrent entre autres de réinitialiser les poids dès que des descentes ont eu lieu, et d'incrémenter le poids des clauses d'un facteur donné en paramètre de la procédure. T. Castell et M. Cayrol dans [Castell & Cayrol 1996a] utilisent une version de Break Out Method intégrant un système de « saut » et d'interprétation miroir pour illustrer leur propos sur la génération d'instances aléatoires (cf. page 51). Ils étendent le voisinage avec l'interprétation miroir qui n'est autre que l'interprétation complémentaire de l'interprétation courante et proposent de flipper l'ensemble des variables apparaissant dans les clauses falsifiées (« saut ») lorsque l'algorithme a atteint un optimum local. Dans le même article [Castell & Cayrol 1996a] une seconde version de l'algorithme de P. Morris est présentée, dans laquelle l'interprétation miroir est choisie comme nouvelle interprétation courante lorsque l'on a atteint un minimum local (cette version n'étend pas le voisinage).

6.1.5 WSAT et ses variantes

WSAT ou « *WalkSAT* » est l'héritier direct de GSAT. Cet algorithme proposé par Selman, Kautz et Cohen [Selman *et al.* 1994] peut être vu comme une nouvelle adaptation de « *Random Walk Strategy* » où un procédé aléatoire n'est plus seulement utilisé pour s'extraire des optima locaux mais aussi comme un moyen de diminuer le nombre d'interprétations voisines.

En effet, dans WSAT le voisinage est un sous-ensemble du voisinage direct utilisé par GSAT. Dans cette méthode, une clause falsifiée est choisie aléatoirement et c'est parmi les variables de cette clause que la variable à réparer est élue. WSAT définit donc un voisinage extrêmement réduit, qui accélère notablement les temps de calcul. La fonction voisinage de WSAT est une fonction à deux paramètres : l'interprétation courante I et une clause c falsifiée par I et se définit de la façon suivante :

Définition 6.2 (Voisinage de WSAT)
$Voisins_{wsat}(I, c) = \{J \mid J$ est une interprétation t.q. $\delta_{H(I,J)} = \{x\}$ avec x une variable appartenant à $c\}$

La fonction d'évaluation reste « *min-conflict* », à savoir que la variable réparée est celle qui permet de réduire au maximum, le nombre de clauses falsifiées. Comme dans GSAT, l'exploration des plateaux et les remontées sont autorisées. L'algorithme 6.8 ci-dessous décrit cette procédure.

Algorithme 6.8 WSAT

```
1.  Function WSAT : boolean
2.  Input  : Σ un ensemble de clauses, deux entiers : Max_Tries et Max_Flips ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     for i from 1 to Max_Tries
6.     do
7.         I=une interprétation complète générée aléatoirement ;
8.         for j from 1 to Max_Flips
9.         do
10.           if (I est un modèle)
11.           then
12.               return true ;
13.           else
14.              c=une clause au hasard parmi les clauses falsifiées par I ;
15.              foreach variable v de c
16.              do
17.                 Calculer score[v] ;          %% le score utilisé est min-conflict
18.              done
19.              list_of_max_diff=liste des variables ayant le meilleur score ;
20.              x=une variable au hasard parmi list_of_max_diff ;
21.              I=(I avec la valeur d'affectation de x inversée) ;
22.           fi
23.        done
24.    done
25.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
26. End
```

Les différentes optimisations mises au point pour GSAT, peuvent aisément se greffer à WSAT. Nous restreignons notre présentation à l'adaptation de « *Random Walk Strategy* » pour WSAT et aux deux nouvelles stratégies qui sont actuellement reconnues comme les plus efficaces [McAllester *et al.* 1997].

6.1.5.1 Noise

L'adaptation de « *Random Walk Strategy* » à WSAT est immédiate :

- avec la probabilité p on répare une variable au hasard dans la clause falsifiée choisie préalablement ;
- avec une probabilité $1 - p$ on répare la variable possédant le meilleur score parmi les variables de la clause.

Dans l'algorithme de WSAT (cf. 6.8 page précédente), il suffit de remplacer la ligne 20 par le pseudo-code suivant :

Algorithme 6.9 WSAT+NOISE

```
...    ...                                                    %% Lignes 1 à 19 de WSAT
20.    n=un nombre généré aléatoirement entre 0 et 1 ;
20.a   if (p > n)
20.b   then                                                                 %% Noise
20.c     x=une variable au hasard parmi les variables de c;
20.d   else                                          %% WSAT classique : min-conflict
20.e     x=une variable au hasard parmi list_of_max_diff ;
20.f   fi
...    ...                                                    %% Lignes 21 à 26 de WSAT
```

Cette stratégie est appelée « *noise* » dans l'implantation originelle de WSAT.

Il faut noter que si l'on fixe p à 1, cet algorithme est présenté dans [Papadimitriou 1991]. C.H. Papadimitriou prouve dans le même article que de manière probabilite cet algorithme permet de résoudre en temps quadratique les instances *2***SAT** ; ce résultat ayant essentiellement un intérêt théorique.

Théorème 6.1 ([Papadimitriou 1991])
*Soit Σ une instance consistante 2***SAT** *construite sur n variables. Si* `Max_Tries` $> 2n^2$ *alors la probabilité pour que WSAT+Noise (avec $p = 1$) trouve un modèle est supérieure à un demi.*

6.1.5.2 Novelty

Dans la stratégie « *Novelty* », les variables (de la clause falsifiée tirée aléatoirement) sont triées en fonction de leur score (toujours « *min-conflict* »). Un critère d'ancienneté permet par la suite de désigner la variable à réparer. En effet, dans cette stratégie, chaque variable se voit associé un entier (`level`) indiquant à quel moment remonte son dernier flip.

Lors de l'élection de la variable à flipper, les deux variables arrivées en tête lors du tri, sont considérées. Si la meilleure de ces deux variables est la moins récemment flippée, c'est elle qui est choisie, sinon avec

une probabilité p, on choisit la seconde variable et avec la probabilité complémentaire on choisit la meilleure variable.

L'algorithme 6.10 ci-dessous décrit en détail l'algorithme WSAT+Novelty.

Algorithme 6.10 WSAT+NOVELTY

```
1.  Function GSAT+Novelty : boolean
2.  Input  : Σ un ensemble de clauses, deux entiers : Max_Tries et Max_Flips,
             une probabilité p de réparer la variable la moins récemment flippée ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     for i from 1 to Max_Tries do
6.         I=une interprétation complète générée aléatoirement ;
7.         foreach variable in Σ do
8.            level[v]=0 ;
9.         done
10.        for j from 1 to Max_Flips do
11.           if (I est un modèle) then return true ;
12.           else
13.              c=une clause au hasard parmi les clauses falsifiées par I ;
14.              foreach variable v de c do
15.                 Calculer score[v] ;          %% le score utilisé est min-conflict
16.              done
17.              v_best=la variable ayant le meilleur score ;
18.              v_second=la variable ayant le deuxième meilleur score ;
19.              if (score[v_best]==score[v_second]) then
20.                 if (level[v_best] > level[v_second])
21.                 then x=v_second ;
22.                 else x=v_best ;
23.                 fi
24.              elif (level[v_best] < level[v_second])
25.              then x=v_best ;
26.              else
27.                 n=un nombre généré aléatoirement entre 0 et 1 ;
28.                 if (p > n)
29.                 then x=v_second ;
30.                 else x=v_best ;
31.                 fi
32.              fi
33.              I=(I avec la valeur d'affectation de x inversée) ;
34.              level[x]=j ;
35.           fi
36.        done
37.    done
38.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
39. End
```

Le critère d'ancienneté utilisé permet d'éviter des flips trop récurrents. Il peut être assimilé à une sorte de tabou, dans le sens où on minimise le risque qu'une variable récemment flippée le soit de nouveau.

En fixant la probabilité pour Noise à 0.6, la combinaison de Noise et Novelty semble, sur la base de tests étudiés par McAllester *et al.*, améliorer très nettement les performances de WSAT [McAllester *et al.* 1997]. Cependant à notre connaissance, aucun réglage du paramètre p pour la procédure Novelty n'a été proposé, ce qui rend la stratégie difficilement exploitable.

6.1.5.3 R-Novelty

La stratégie « *R-Novelty* » est un raffinement de « *Novelty* ». Comme pour WSAT+Novelty, les variables possédant les deux meilleurs scores sont étudiées. Les stratégies sont identiques sauf lorsque la meilleure variable (`v_best`) est la plus récemment flippée. Dans ce cas, la différence d entre les scores des deux meilleures variables (`v_best` et `v_second`) est considérée. Quatre cas sont distingués :

1. $p < 0,5$ et $d > 1$: `v_best` est choisie ;
2. $p < 0,5$ et $d = 1$: avec la probabilité $2p$, `v_second` est choisie et avec la probabilité $1 - 2p$, `v_best` est choisie ;
3. $p \geqslant 0,5$ et $d = 1$: `v_second` est choisie ;
4. $p \geqslant 0,5$ et $d > 1$: avec la probabilité $2(p - 0,5)$, `v_second` est choisie et avec la probabilité $1 - 2(p - 0,5)$, `v_best` est choisie.

Les modifications à effectuer à l'algorithme WSAT+Novelty (cf. algorithme 6.10 page précédente), pour obtenir celui de WSAT+R-Novelty portent sur les lignes comprises entre 27 et 32. Ces lignes doivent être remplacées par le pseudo-code ci-dessous.

Algorithme 6.11 WSAT+R-NOVELTY

```
...     ...                                              %% Lignes 1 à 26 de WSAT+Novelty
27.     d=score[v_best]-score[v_second] ;
28.     if ((p < 0.5) && (d > 1)) then x=v_best ;                          %% Cas 1
29.     elif ((p < 0.5) && (d == 1)) then                                  %% Cas 2
29.a      n un nombre généré aléatoirement entre 0 et 1 ;
29.b      if ((2*p) > n )
29.c      then x=v_second ;
29.d      else x=v_best ;
29.e      fi
30.     elif ((p >= 0.5) && (d = 1)) then x=v_second ;                     %% Cas 3
31.     else                                          %% Cas 4 : ((p >= 0.5) && (d > 1))
31.a      n un nombre généré aléatoirement entre 0 et 1 ;
31.b      if ((2*(p-0.5)) > n )
31.c      then x=v_second ;
31.d      else x=v_best ;
31.e      fi
31'.    fi
...     ...                                              %% Lignes 32 à 39 de WSAT+Novelty
```

McAllester *et al.* dans leur article [McAllester *et al.* 1997] proposent de mixer cette approche avec la stratégie Noise, en fixant la probabilité de bruit à $0,6$. Cette technique (WSAT+R-Novelty+Noise(p=0,6)) semble fournir de meilleurs résultats comparativement aux autres approches basées sur WSAT. Cependant,

aucune précision n'est donnée quant à la valeur de la probabilité à utiliser pour R-Novelty. De plus, McAllester *et al.* affirment que le réglage de ce paramètre est très délicat et que les performances de l'algorithme sont très dépendantes de ce réglage.

6.2 TSAT

L'aspect aléatoire prédominant dans GSAT, WSAT et leurs variantes est une source de difficultés pour l'analyse et la compréhension de l'efficacité de ces méthodes. Cette remarque nous a amenés à nous poser la question suivante : « *Dans quelle mesure est-il possible de trouver une méthode conservant des performances équivalentes aux meilleures approches connues, tout en atténuant au maximum l'aspect aléatoire qui guide les réparations ?* ».

Dans [Selman *et al.* 1992], B. Selman affirme que : « *An other feature of GSAT is that the variable whose assignment is to be changed is chosen at random from those that would give an equally good improvement. Such non-determinism makes it very unlikely that the algorithm makes the same sequence of changes over and over.* ». Cependant, après une analyse expérimentale détailléee de GSAT+RWS, sur des problèmes consistants pour lesquels la recherche aboutit à un optimum local, nous avons constaté que GSAT+RWS flippe souvent les mêmes variables. Ces réparations redondantes amènent l'algorithme à effectuer des cycles dans ces réparations.

De ces constatations, nous proposons une nouvelle méthode, appelée TSAT [Mazure *et al.* 1997e][6] : (« *Tabu for SAT* »). Cette méthode se base sur la même ossature que GSAT en effectuant une utilisation systématique d'une liste de réparations interdites pour « éviter »les flips récurrents et échapper aux minima locaux. Ceci permet d'explorer et de couvrir au mieux l'espace de recherche.

Préalablement à la description détaillée de l'algorithme, nous décrivons les différents choix possibles pour confectionner un algorithme à base de tabou.

6.2.1 Les « différents » Tabou

Le principe de la liste tabou (cf. paragraphe 6.1.3 page 59) consiste à interdire les déplacements conduisant à un ensemble d'interprétations déjà visitées et caractérisées par le contenu de la liste tabou [Glover 1989, Glover 1990]. Il existe différentes manières de gérer cette liste tabou. Lors de la réalisation d'une méthode de recherche locale à base de tabou, certaines décisions doivent être prises quant à la gestion et à l'utilisation de la liste tabou [Glover *et al.* 1993, Glover & Laguna 1993].

Nous regroupons ces choix en trois catégories :

1. contenu de la liste ;
2. longueur de la liste ;
3. règles d'application du tabou :
   - moment d'utilisation de la liste ;
   - critère d'aspiration.

6. Une version préliminaire de ce papier est parue dans [Mazure *et al.* 1995]. Dans le papier initial, TSAT est appelé TWSAT (« *Tabu Walk Strategy for SAT* »). De nombreuses confusions, nous ont conduit à renommer cet algorithme. En effet, le « W » de TWSAT semblait faire allusion pour beaucoup de lecteurs à celui de WSAT. Or l'ossature de TSAT (TWSAT) est plus proche de celle de GSAT que de celle de WSAT.

Le contenu de la liste tabou va représenter les interprétations voisines inaccessibles. Dans le cas de **SAT**, comme les déplacements entre les interprétations se font par l'intermédiaire des variables, trois objets sont susceptibles d'appartenir à la liste d'interdits : les interprétations, les variables elles-mêmes ou les clauses. Dans le cas des interprétations, la liste tabou caractérise l'ensemble des interprétations ne pouvant pas être atteintes lors de la prochaine réparation. Le principe de tabou étant d'éviter de repasser par des interprétations déjà visitées, une liste tabou contenant des interprétations semblent donc idéale. Cependant en pratique, la mise en œuvre d'une telle liste tabou risque d'être coûteuse en mémoire (sauvegarde d'une liste d'interprétations) mais également en temps (comparaisons entre les interprétations voisines de l'interprétation courante et celles contenues dans la liste tabou). En effet, dans certains cas la liste d'interdits peut être longue et donc de nombreuses interprétations doivent être mémorisées. Une alternative consiste à stocker dans la liste les variables récemment réparées. Dans ce cas, la liste tabou caractérise l'ensemble des interprétations ne pouvant être atteintes au prochain flip et qui diffèrent de l'interprétation courante par au moins une des variables de la liste. Cette approche semble fournir un juste compromis et c'est elle qui sera employée dans TSAT. Enfin la dernière possibilité consiste à mémoriser la trace des clauses réparées. L'idée est de mémoriser chaque clause venant d'être réparée, c'est-à-dire les clauses qui sont passées d'un état de contradiction à celui de satisfaction. Le tabou est utilisé dans ce cas, comme un moyen d'éviter de réparer toujours les mêmes clauses afin de mieux couvrir l'espace de recherche. Dans ce cas, la liste tabou caractérise l'ensemble des interprétations qui ne peuvent être atteintes au prochain flip et qui diffèrent par leurs ensembles respectifs de clauses falsifiées.

La seconde question à résoudre porte sur la longueur de la liste tabou. Un autre paramètre important caractérisant les méthodes à base de tabou concerne la longueur de la liste tabou et la manière dont elle évolue au cours de la recherche. Trois types de tailles peuvent être envisagées : fixe, dynamique ou illimité. Lorsque la taille de la liste tabou est fixe, elle est imposée au début et reste constante tout au long de la recherche. Cette stratégie est celle que l'on rencontre le plus fréquemment dans les algorithmes de recherche locale à base de tabou. C'est cette option qui est implantée dans TSAT. Une autre approche consiste à faire évoluer la taille de la liste tabou au cours de la recherche. On dit alors que l'on utilise une liste tabou dynamique. Cette stratégie doit logiquement fournir de meilleurs résultats que la précédente. En effet, il y a de fortes chances pour que la taille de la liste tabou ne soit pas identique, que l'on soit dans une phase de descente, d'exploration de plateau ou de remontée. Le problème de cette stratégie est de déterminer quand et de quelle manière faire évoluer la liste tabou. Dans la section 6.2.5 nous décrivons une variante de TSAT utilisant une liste tabou dynamique. La dernière alternative concernant la taille de la liste tabou est d'utiliser une liste de longueur illimitée. Cette technique n'a de sens que si un tabou sur les interprétations est employé. Cependant une telle stratégie est utopique tant l'espace mémoire requis est conséquent. Par ailleurs, une telle méthode revient à effectuer une exploration systématique de l'espace des interprétations.

Une fois le contenu et le type de longueur de la liste tabou fixés, il reste à déterminer les conditions d'utilisation de cette liste. Elle peut être utilisée tout au long de la recherche ou uniquement lorsqu'un optimum local est atteint. Une utilisation restreinte aux optima locaux signifie que lorsqu'une interprétation dans le voisinage permet une descente, la configuration courante est réparée vers cette interprétation que cette interprétation soit ou non interdite par le contenu de la liste tabou. À l'inverse, l'utilisation systématique de la liste tabou signifie qu'une interprétation interdite par le contenu de la liste tabou, ne sera jamais choisie même si elle est la seule à permettre une descente. C'est cette utilisation qui a été choisie pour TSAT. Nous pensons qu'une telle manière de procéder permet d'éviter certains minima locaux. De manière imagée, cette utilisation peut être vue comme « un serpent se mouvant dans un paysage montagneux et ne s'aventurant dans une crevasse que si celle-ci est assez profonde pour accueillir sur sa pente la totalité du corps du reptile ».

Par ailleurs, il est possible de greffer à ces deux types d'utilisation de la liste tabou (systématique ou restreinte aux optima locaux), un principe additionnel appelé « critère d'aspiration » [Glover 1989, Glover 1990]. Ce critère autorise la violation du principe tabou lorsque le déplacement vers une interprétation voisine améliore la meilleure valeur de la fonction d'évaluation obtenue depuis le début de la recherche. C'est-à-dire que si parmi le voisinage une interprétation donne une valeur de la fonction d'éva-

luation meilleure que toutes celles obtenues par les interprétations déjà visitées, alors la réparation vers cette interprétation est effectuée que cette interprétation soit ou non interdite par la liste tabou. À notre connaissance, toutes les méthodes à base de tabou pour **SAT** utilisent ce critère d'aspiration. Cependant afin de ne pas perturber les déplacements de notre serpent, TSAT n'utilise pas ce principe. De plus, le réglage optimal de la longueur (cf. paragraphe 6.2.3 page 73) de la liste tabou est plus délicat lorsque le critère d'aspiration est appliqué.

La table 6.1 synthétise les diverses possibilités offertes lors de la réalisation d'un algorithme de recherche locale à base de tabou. La sélection d'une case à chaque ligne de cette table permet d'obtenir une méthode tabou spécifique. Les choix faits lors de la réalisation de TSAT sont indiqués en gras dans cette table.

Paramètres	Choix possibles		
Contenu de la liste tabou	Interprétations	**Variables**	Clauses
Type de longueur de la liste tabou	**Fixe**	Dynamique	Illimitée
Utilisation du tabou	**Systématique**	Restreinte aux minima locaux	
Critère d'aspiration	Avec	**Sans**	

TAB. 6.1 – *Les différents paramètres d'une méthode tabou*

6.2.2 L'algorithme

TSAT gère une liste FIFO, de longueur fixe, des variables les plus récemment flippées. Après chaque flip, la variable ayant été réparée entre dans cette liste d'interdits où elle y restera durant un certain nombre d'itérations correspondant à la longueur de la liste. La différence majeure, avec la méthode tabou classique réside dans le fait que cette liste est utilisée tout au long de la recherche et non pas seulement lorsqu'une détérioration de la configuration courante doit être recherchée. C'est-à-dire, tant qu'une variable est dans la liste tabou, elle ne peut participer à la réparation même si son inversion provoque l'évolution dans le sens de la progression la plus forte.

La fonction d'évaluation des interprétations reste identique à celle de GSAT ou WSAT, à savoir « *Min-conflict* ». Nous rappelons que cette fonction a pour valeur le nombre de clauses falsifiées.

Une différence entre TSAT et les méthodes jusqu'alors présentées réside dans le voisinage. Afin de minimiser le temps d'évaluation du voisinage de l'interprétation courante, TSAT utilise comme voisinage une restriction du voisinage direct (cf. définition 6.1) :

Définition 6.3 (Voisinage de TSAT)
$Voisins_{TSAT}(I) = \{J \mid J$ est une interprétation t.q. $\delta_{H(I,J)} = \{v\}$ avec v une variable non tabou et appartenant à au moins une clause (de Σ) falsifiée par $I\}$.

La variable réparée est donc choisie pami l'ensemble des variables apparaissant dans l'ensemble des clauses falsifiées de l'interprétation courante. Le choix porte sur la variable qui donne la plus forte progression (ou la plus faible régression) de la fonction objective et qui n'appartient pas à la liste tabou. Dans le cas où toutes les variables apparaissant dans des clauses falsifiées appartiennent également à la liste tabou, TSAT autorise la réparation de la meilleure variable de la liste tabou.

Par ailleurs, un choix aléatoire est effectué parmi les variables possédant le meilleur score de la fonction d'évaluation. De plus, l'interprétation initiale est générée aléatoirement. Une description détaillée de TSAT est faite dans l'algorithme ci-dessous.

Algorithme 6.12 TSAT

```
1.  Function TSAT : boolean
2.  Input  : Σ un ensemble de clauses, trois entiers : Max_Tries, Max_Flips et l
             la longueur de la liste tabou  ;
3.  Output : true si Σ admet un modèle, false s'il est impossible de conclure ;
4.  Begin
5.     list_tabou=new liste FIFO de longueur l ;
6.     for i from 1 to Max_Tries do
7.         I=une interprétation complète générée aléatoirement ;
8.         liste_tabou=∅ ;
9.         for j from 1 to Max_Flips do
10.           if (I est un modèle) then return true ;
11.           else
12.             foreach variable v appartenant à des clauses de Σ falsifiées
                        par I et t.q. v ∉ liste tabou do
13.                fals_to_sat[v]=le nombre de clauses falsifiées par I qui
                                  deviendraient satisfaites, si v était flippée ;
14.                sat_to_fals[v]=le nombre clauses satisfaits par I qui
                                  deviendraient falsifiées, si v était flippée ;
15.                score[v]=fals_to_sat[v]-sat_to_fals[v] ;        %% min-conflict
16.             done
17.             liste_best=liste des variables ayant le meilleur score ;
18.             if (liste_best==∅) then  %% Toutes les variables falsifiées sont tabou
19.                foreach variable v appartenant à liste tabou do
20.                   fals_to_sat[v]=le nombre de clauses falsifiées par I qui
                                     deviendraient satisfaites, si v était flippée ;
21.                   sat_to_fals[v]=le nombre clauses satisfaits par I qui
                                     deviendraient falsifiées, si v était flippée ;
22.                   score[v]=fals_to_sat[v]-sat_to_fals[v] ;     %% min-conflict
23.                done
24.                liste_best=liste des variables ayant le meilleur score ;
25.             fi
26.             x=une variable au hasard parmi liste_best ;
27.             I=(I avec la valeur d'affectation de x inversée) ;
28.             mettre à jour liste_tabou ;   %% x entre dans la liste et la variable
                                              %% réparée l flips auparavant sort
29.          fi
30.       done
31.    done
32.    return (I est modèle) ; %% Retouner la valeur du test : « I est-il un modèle ? »
33. End
```

6.2.3 Réglage des paramètres

Les paramètres des méthodes de recherche locale jouent un rôle fondamental, leur réglage influence considérablement leur efficacité. Par exemple, B. Selman *et al.* [Selman *et al.* 1993] proposent un réglage pointu des paramètres de GSAT en particulier de la probabilité p pour « *Random Walk Strategy* », cette probabilité influe énormément sur les performances de l'algorithme et diffère suivant la nature des instances. Par ailleurs, plusieurs auteurs (*e.g.* [Gent & Walsh 1993a, Parkes & Walser 1996] ont étudié les performances de GSAT en fonction des paramètres `Max_Flips` et `Max_Tries`. En ce qui concerne TSAT le paramètre critique est la longueur de la liste d'interdits. Un bon réglage de ce paramètre est donc primordial.

Afin de déterminer la taille optimale de la liste tabou, nous avons effectué des séries intensives d'expérimentations. Nous nous sommes tout particulièrement intéressés au réglage de TSAT pour les instances *k***SAT** aléatoires.

Nous avons généré des instances *3***SAT** au seuil ($C/V = 4,25$) dont le nombre de variables varie de 50 à 1500 par pas de 50. Pour chaque taille de problèmes nous avons testé un nombre significatif d'instances (500 pour les problèmes de 50 à 1000 variables, 200 pour les problèmes de taille supérieure à 1000).

Pour chaque instance nous avons expérimenté TSAT avec une longueur de liste tabou variant de 1 à 50. La figure 6.1 montre la longueur optimale de la liste tabou en fonction du nombre de variables.

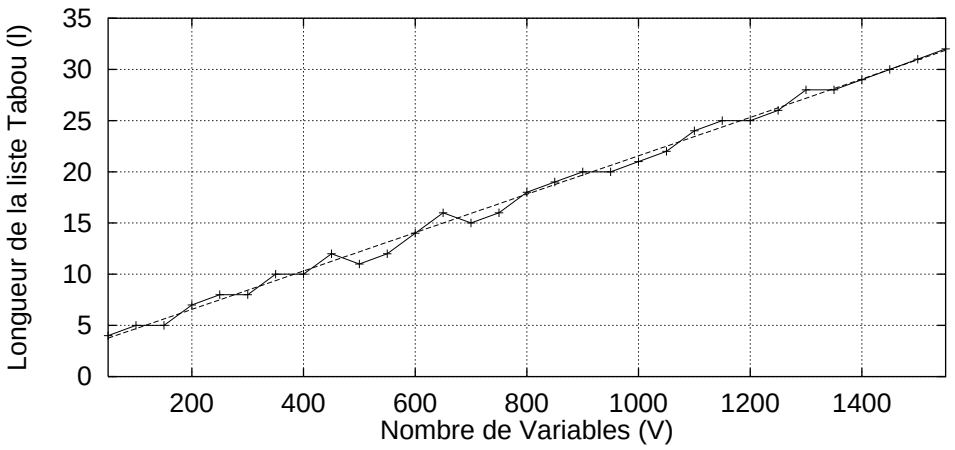

FIG. 6.1 – *Taille optimale de la liste tabou de TSAT pour des instances 3***SAT** *aléatoires au seuil*

Sur l'intervalle considéré, la courbe apparaît comme une fonction linéaire du nombre de variables. De plus, nous avons constaté que :

- plus la longueur de la liste tabou s'éloigne de la longueur optimale, plus les performances se dégradent ;
- cette longueur optimale reste identique pour des instances générées en dehors du seuil ;
- la taille de la liste tabou ne semble pas dépendre de la taille des clauses, ceci a été observé en conduisant le même type d'expérimentations, à moindre échelle, sur des instances *4***SAT** .

Expérimentalement, la longueur de la liste tabou pour des instances *k***SAT** aléatoires est donnée par la fonction linéaire suivante :

$$\boxed{l = 0,01875 \times V + 2,8125}$$

Ce résultat étonnant mériterait une étude analytique plus approfondie, ceci afin de comprendre la nature des instances *k***SAT** aléatoires et d'en faire éventuellement émerger de nouvelles propriétés. Intuitivement, il nous semble qu'il pourrait être possible de lier cette courbe à la hauteur moyenne des optima locaux. Par ailleurs, nous avons pu prouver expérimentalement que cette taille correspond à la taille des cycles de réparations pour les instances *k***SAT** (cf. paragraphe 6.2.5).

6.2.4 Résultats expérimentaux

Nous présentons une comparaison expérimentale entre GSAT+RWS, WSAT+Noise et TSAT. L'algorithme TSAT est implanté en C sur une plate-forme incluant divers algorithmes [Mazure *et al.* 1996d, Mazure *et al.* 1998b]. En ce qui concerne GSAT+RWS et WSAT+Noise les implantations originelles de leurs auteurs ont été utilisées. Les expérimentations ont été menées sur des PC sous le système d'exploitation Linux.

6.2.4.1 Instances aléatoires

Les différentes approches ont été évaluées sur des instances *3***SAT** au seuil. La comparaison porte sur plusieurs critères :

- le temps moyen (exprimé en secondes) nécessaire à la résolution d'une instance ;
- le nombre de réparations effectuées en moyenne pour résoudre une instance ;
- le pourcentage de problèmes résolus[7].

Nous avons également indiqué le rapport du nombre moyen de réparations sur le pourcentage de problèmes résolus. Ceci afin d'obtenir une comparaison plus équitable lorsqu'un algorithme résout plus d'instances que les autres.

Les valeurs des paramètres `Max_Flips` et `Max_Tries`, communs aux trois approches, sont celles fournies par Andrew J. Parkes et Joachim P. Walser dans [Parkes & Walser 1996], c'est-à-dire V^2 flips et 5 « tries » où V représente le nombre de variables de l'instance. Les probabilités utilisées par GSAT+RWS et WSAT+Noise sont celles par défaut lors de l'exécution respective de ces algorithmes (à savoir $0,5$ pour les deux approches). En ce qui concerne TSAT, la longueur de la liste d'interdits est calculée par l'équation donnée précédemment (cf. paragraphe 6.2.3).

La table 6.2 page ci-contre synthétise les résultats obtenus par GSAT+RWS et TSAT.

À la lecture de cette table plusieurs conclusions peuvent être tirées :

- TSAT est nettement plus efficace que GSAT+RWS sur les instances $3\mathbf{SAT}$ aléatoires au seuil ;
- plus le nombre de variables augmente, plus le gain obtenu par TSAT s'accentue. Ceci est illustré par la courbe suivante (cf. FIG. 6.2).

Dans un second temps le même type d'expériences a été réalisé avec WSAT+Noise, nous présentons dans la courbe 6.3 les résultats obtenus par les trois approches (WSAT+Noise, GSAT+RWS et TSAT). Cette courbe exprime comme précédemment le ratio du nombre moyen de flips sur le pourcentage d'instances résolues en fonction du nombre de variables. Pour chaque nombre de variables, variant entre 100 et 2000 (par

7. Le pourcentage des problèmes résolus est relatif au 50% de problèmes estimés satisfaisables au seuil.

problèmes		Nb.	GSAT+RWS				TSAT			
V	C	inst.	temps	#flips	résolus	ratio	temps	#flips	résolus	ratio
100	430	500	.18	2803	88%	31.85	.11	1633	93%	17.60
200	860	500	1.99	18626	73%	255.85	.73	9678	74%	130.78
400	1700	500	15.03	204670	100%	2046.70	11.51	145710	100%	1457.10
600	2550	500	19.59	250464	62%	4013.85	13.92	167236	65%	2580.80
800	3400	500	140.61	1809986	67%	26854.39	99.45	1143444	71%	16150.34
1000	4250	500	369.88	4633763	57%	81009.84	292.10	323246 3	62%	51802.29
2000	8240	50	3147.26	26542387	16%	1658899.19	3269.15	294154 65	40%	735386.63

TAB. 6.2 – *TSAT vs. GSAT+RWS*

pas de 100 variables), 200 instances ont été générées. Par ailleurs, contrairement aux premières expériences, nous avons limité le temps d'exécution des algorithmes (traiter un volume non négligeable d'instances de plus de 1000 variables au seuil est une opération très coûteuse en temps CPU, malgé l'efficacité croissante des méthodes de recherche locale).

Cette courbe confirme l'efficacité de TSAT comparativement à GSAT+RWS et même à WSAT+Noise (pour les instances de taille inférieure à 500 variables). Cependant, dès lors que le nombre de variables devient important, WSAT+Noise s'affirme comme la méthode la plus efficace.

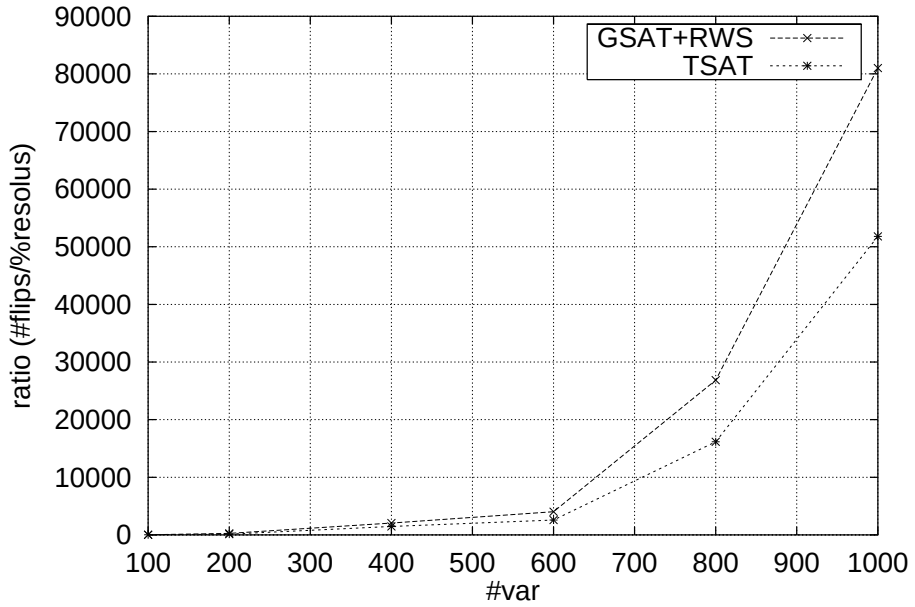

FIG. 6.2 – *GSAT+RWS vs TSAT : ratio du pourcentage de problèmes résolus sur nombre de flips en fonction du nombre de variables pour des instances 3***SAT** *aléaoires*

Pour conclure sur les instances aléatoires, TSAT est plus efficace que GSAT+RWS, mais est moins performant que WSAT+Noise. Il faut relativiser cette contre-performance vis-à-vis de WSAT+Noise. En effet, Par sa conception, WSAT autorise, dans un temps limité, un nombre de réparations plus important que TSAT (ou GSAT+RWS). Lors du choix de la variable à flipper, seules quelques variables (dans le cas d'instances *3***SAT**, 3 exactement) sont examinées, alors que pour TSAT l'ensemble des variables apparaissant dans des clauses insatisfaites est pris en considération. Ainsi, le temps limite imposé par des contraintes matérielles, joue en défaveur de notre algorithme. Il reste cependant que pour un temps imparti, WSAT+Noise permet de résoudre plus d'instances que TSAT.

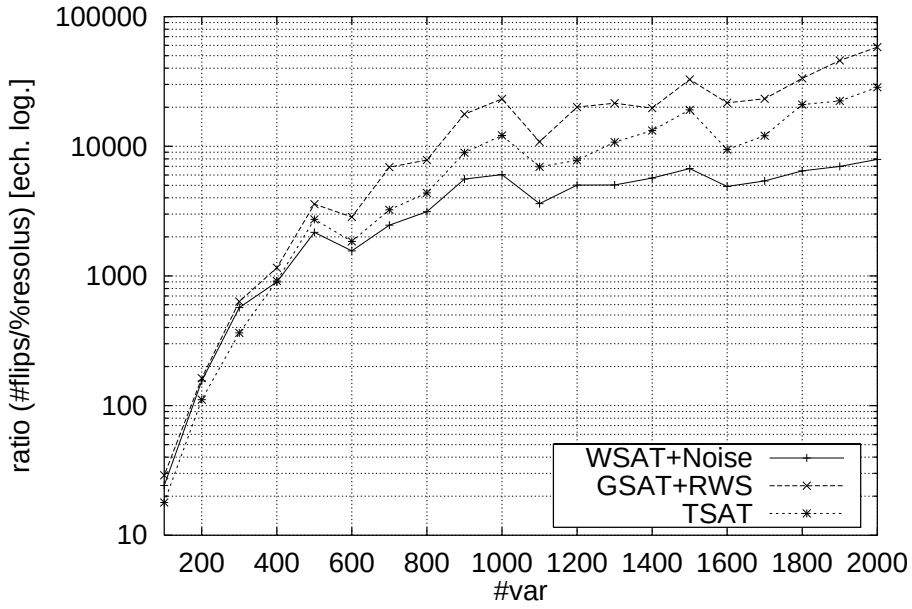

FIG. 6.3 – *WSAT+Noise vs GSAT+RWS vs TSAT : ratio du pourcentage de problèmes résolus sur le nombre de flips en fonction du nombre de variables pour des instances 3***SAT** *aléatoires*

6.2.4.2 Instances structurées

Le second type d'expérimentations réalisé porte sur l'ensemble des instances satisfaisables proposées au challenge DIMACS [DIM1993]. Une description détaillée de ces instances est fournie en annexe 13.

Pour chaque type d'instances, les 3 techniques de réparations locales GSAT+RWS, WSAT+Noise et TSAT ont été testées. Les ressources (nombre de tries et nombre de flips) attribuées à chaque algorithme (identiques pour chaque algorithme) varient suivant le type d'instances. La table ci-dessous indique pour chaque type de problème le nombre de flips attribué ($20 \times V$ ou V^2, V étant le nombre de variables de l'instance) ; le nombre de tries a été fixé à 5 quels que soit l'instance et l'algorithme.

Types d'instances	SSA	F	II	HANOI	PAR	G	AIM	FAW	JNH
Nombre de flips	$20 \times V$	V^2	$20 \times V$	V^2	$20 \times V$	$20 \times V$	V^2	V^2	V^2

TAB. 6.3 – *Conditions expérimentales pour chaque type d'instances de DIMACS*

Par ailleurs, la probabilité de flipper aléatoirement une variable pour GSAT+RWS et WSAT+Noise a été fixée suivant les recommandations de leurs auteurs à 0.5 [Selman *et al.* 1994]. En ce qui concerne la longueur de la liste tabou pour TSAT, l'absence de réglage automatique, nous a obligé à déterminer manuellement la taille optimale de la liste pour chaque instance. Cette taille (lg) est reportée dans les tables pour chacune des instances testées. Enfin, nous avons regroupé dans une même table les instances d'une même catégorie.

Les tables suivantes donnent le nombre de tries et de flips nécessaires dans chaque try pour résoudre chaque instance satisfaisable de DIMACS. Nous avons également précisé les temps de calcul obtenus sur

un PC équipé d'un Pentium II 350Mhz et fonctionnant sous Linux.

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
ssa7552-001	1534	3614		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-002	1534	3614		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-021	1534	3614		*Non résolu*			*Non résolu*		1	18658	0s31	64
ssa7552-032	1530	3605		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-037	1501	3575		*Non résolu*		4	13631	1s21		*Non résolu*		—
ssa7552-038	1501	3575		*Non résolu*		1	22671	0s36		*Non résolu*		—
ssa7552-039	1483	3470		*Non résolu*		3	28603	1s00	1	14027	0s26	158
ssa7552-040	1496	3449		*Non résolu*		5	14311	1s56		*Non résolu*		—
ssa7552-042	1528	3598		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-043	1507	3629		*Non résolu*			*Non résolu*		2	9607	0s70	62
ssa7552-044	1514	3501		*Non résolu*		4	19909	1s23	3	7324	1s21	115
ssa7552-045	1503	3586		*Non résolu*			*Non résolu*		5	18824	2s30	51
ssa7552-046	1503	3586		*Non résolu*		2	5647	0s50	3	24603	1s30	87
ssa7552-047	1496	3449		*Non résolu*			*Non résolu*		1	16025	0s30	146
ssa7552-048	1498	3460		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-049	1498	3458		*Non résolu*		2	20068	0s61		*Non résolu*		—
ssa7552-050	1502	3466		*Non résolu*			*Non résolu*		5	12905	2s23	75
ssa7552-051	1502	3466		*Non résolu*		2	14976	0s58	1	16275	0s30	56
ssa7552-052	1496	3456		*Non résolu*			*Non résolu*		2	25893	1s05	48
ssa7552-053	1501	3598		*Non résolu*		1	6118	0s20		*Non résolu*		—
ssa7552-054	1448	3300		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-055	1473	3442	5	28022	16s95	1	6169	0s18	3	12315	1s10	112
ssa7552-058	1530	3622		*Non résolu*			*Non résolu*		2	20030	0s96	94
ssa7552-059	1512	3523		*Non résolu*		3	16029	0s86	1	13177	0s26	115
ssa7552-060	1494	3446		*Non résolu*			*Non résolu*		1	4625	0s13	157
ssa7552-061	1503	3586		*Non résolu*			*Non résolu*		1	23096	0s40	152
ssa7552-062	1503	3586	5	15744	15s85	5	16006	1s60	4	13474	1s71	116
ssa7552-063	1446	3289		*Non résolu*		1	7778	0s18	5	5868	2s21	115
ssa7552-064	1501	3579	2	25579	9s40		*Non résolu*			*Non résolu*		—
ssa7552-065	1501	3579		*Non résolu*		1	22156	0s35		*Non résolu*		—
ssa7552-066	1481	3463		*Non résolu*		5	22055	1s58	2	6432	0s60	53
ssa7552-067	1448	3300		*Non résolu*		2	17807	0s58	1	20578	0s30	60
ssa7552-074	1480	3458		*Non résolu*			*Non résolu*		5	27541	2s30	57
ssa7552-075	1446	3282		*Non résolu*		1	18745	0s31	2	25808	0s71	82
ssa7552-076	1446	3282		*Non résolu*		4	7072	1s08	2	15061	0s68	63
ssa7552-077	1446	3282		*Non résolu*		4	22127	1s20	1	20762	0s33	103
ssa7552-080	1536	3635		*Non résolu*			*Non résolu*		5	21452	2s41	68
ssa7552-081	1505	3622		*Non résolu*		4	20741	1s30	2	19416	0s78	52
ssa7552-082	1449	3297		*Non résolu*		1	8704	0s20		*Non résolu*		—
ssa7552-083	1448	3298		*Non résolu*		3	11093	0s80		*Non résolu*		—
ssa7552-084	1448	3300		*Non résolu*			*Non résolu*		4	19150	1s65	60
ssa7552-085	1473	3442		*Non résolu*			*Non résolu*		1	14724	0s25	127
ssa7552-090	1517	3665		*Non résolu*		1	9956	0s20	1	29305	0s43	53
ssa7552-091	1449	3297	1	24359	2s16	1	22698	0s35	4	7098	1s81	55
ssa7552-092	1448	3298	5	10841	15s58	3	19233	0s91		*Non résolu*		—
ssa7552-093	1448	3300		*Non résolu*		3	27360	1s01	3	17595	1s18	138
ssa7552-094	1473	3442		*Non résolu*			*Non résolu*		5	28651	2s40	159
ssa7552-095	1461	3331		*Non résolu*		4	18080	1s25		*Non résolu*		—
ssa7552-096	1498	3460		*Non résolu*		4	22873	1s31		*Non résolu*		—
ssa7552-098	1831	4123		*Non résolu*			*Non résolu*			*Non résolu*		—
ssa7552-102	1532	3623		*Non résolu*			*Non résolu*		3	13962	1s30	96
ssa7552-103	1498	3460		*Non résolu*		5	27691	1s63		*Non résolu*		—
ssa7552-104	1498	3458		*Non résolu*		2	28642	0s71	5	17667	2s45	175

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
ssa7552-106	1481	3463	*Non résolu*			*Non résolu*			3	11981	1s11	105
ssa7552-107	1501	3575	*Non résolu*			*Non résolu*			1	25867	0s38	81
ssa7552-108	1483	3470	*Non résolu*			3	9049	0s80	1	7925	0s18	89
ssa7552-109	1481	3463	*Non résolu*			2	8575	0s50	2	13335	0s66	85
ssa7552-110	1481	3463	*Non résolu*			*Non résolu*			2	8809	0s60	83
ssa7552-111	1473	3442	*Non résolu*			1	20050	0s30	2	13007	0s71	70
ssa7552-112	1442	3275	*Non résolu*			2	26829	0s71	*Non résolu*			—
ssa7552-113	1542	3665	*Non résolu*			*Non résolu*			2	16425	0s83	134
ssa7552-114	1498	3460	*Non résolu*			3	21689	0s95	*Non résolu*			—
ssa7552-115	1498	3458	*Non résolu*			1	13974	0s25	*Non résolu*			—
ssa7552-116	1502	3466	*Non résolu*			2	24030	0s66	1	9990	0s18	42
ssa7552-117	1494	3440	*Non résolu*			2	27680	0s73	5	22848	2s28	78
ssa7552-118	1494	3440	*Non résolu*			*Non résolu*			*Non résolu*			—
ssa7552-119	1430	3239	*Non résolu*			*Non résolu*			*Non résolu*			—
ssa7552-121	1512	3523	*Non résolu*			*Non résolu*			1	10484	0s21	151
ssa7552-124	2013	4795	*Non résolu*			2	10184	0s63	*Non résolu*			—
ssa7552-125	1512	3523	*Non résolu*			1	27902	0s38	2	18784	0s78	43
ssa7552-126	1490	3434	*Non résolu*			1	20168	0s31	2	13347	0s70	88
ssa7552-127	1430	3239	*Non résolu*			1	23684	0s35	*Non résolu*			—
ssa7552-128	1454	3347	*Non résolu*			*Non résolu*			*Non résolu*			—
ssa7552-129	1430	3239	*Non résolu*			*Non résolu*			*Non résolu*			—
ssa7552-130	1426	3234	*Non résolu*			*Non résolu*			*Non résolu*			—
ssa7552-131	1440	3302	*Non résolu*			*Non résolu*			5	14361	1s86	86
ssa7552-132	1446	3282	*Non résolu*			2	14974	0s53	2	21859	0s85	97
ssa7552-156	1444	3282	*Non résolu*			2	20312	0s56	3	9615	1s00	75
ssa7552-157	1471	3431	*Non résolu*			4	27757	1s33	2	7850	0s58	91
ssa7552-158	1363	3034	*Non résolu*			2	15050	0s51	1	11048	0s16	148
ssa7552-159	1363	3032	2	23431	5s75	1	20720	0s31	1	5822	0s11	98
ssa7552-160	1391	3126	2	23191	5s66	2	9886	0s48	1	2831	0s08	173

TAB. 6.4 – GSAT+RWS vs WSAT+Noise vs TSAT sur les instances SSA

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
f1000	1000	4250	5	937233	519s51	1	297898	5s81	1	114854	2s31	22
f2000	2000	8500	*Non résolu*			1	862906	17s70	3	1152494	219s00	22
f600	600	2550	2	86475	35s35	1	74788	1s43	1	16880	0s30	9

TAB. 6.5 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances F*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
ii16a1	1650	19368	1	5561	2s25	1	1196	0s50	1	734	0s15	10
ii16a2	1602	23281		*Non résolu*		1	2429	0s70	1	3829	0s26	78
ii16b1	1728	24792	1	18436	6s78	1	1388	0s71	2	20134	4s06	92
ii16b2	1076	16121		*Non résolu*		1	1620	0s48		*Non résolu*		—
ii16c1	1580	16467	1	1374	0s78	1	776	0s50	1	795	0s18	62
ii16c2	924	13803		*Non résolu*		2	17550	3s73	4	3725	2s10	60
ii16d1	1230	15901	1	2107	1s01	1	1291	0s53	1	3379	0s26	55
ii16d2	836	12461	4	8770	15s58	1	10434	1s26		*Non résolu*		—
ii16e1	1245	14766	1	1665	1s20	1	888	0s56	1	609	0s16	22
ii16e2	532	7825	1	10563	3s05	1	1065	0s35	1	7434	0s35	49
ii32a1	459	9212	1	2043	1s23	1	940	0s36	2	2577	1s10	159
ii32b1	228	1374	1	892	0s20	1	326	0s10	1	159	0s01	32
ii32b2	261	2558	1	1775	0s58	1	292	0s11	1	2534	0s11	10
ii32b3	348	5734	1	985	1s08	1	1237	0s33	1	364	0s06	75
ii32b4	381	6918	1	4279	2s83	1	788	0s38	1	1718	0s15	16
ii32c1	225	1280	1	542	0s15	1	125	0s08	1	125	0s00	82
ii32c2	249	2182	1	544	0s23	1	158	0s15	1	141	0s01	37
ii32c3	279	3272	1	2122	1s11	1	751	0s21	1	298	0s03	63
ii32c4	759	20862	1	2569	4s45	1	1283	1s13	1	1183	0s33	144
ii32d1	332	2703	1	3238	0s63	1	382	0s11	1	761	0s03	158
ii32d2	404	5153	1	2630	0s93	1	1268	0s26	1	3427	0s21	166
ii32d3	824	19478	3	7652	26s51	1	2586	0s83	4	13445	3s76	25
ii32e1	222	1186	1	559	0s15	1	116	0s08	1	92	0s00	54
ii32e2	267	2746	1	1199	0s51	1	1420	0s21	1	188	0s01	52
ii32e3	330	5020	1	2051	1s46	1	383	0s23	1	329	0s05	34
ii32e4	387	7106	1	1345	1s35	1	388	0s31	1	187	0s06	14
ii32e5	522	11636	1	3152	3s55	1	1189	0s55	1	600	0s13	84
ii8a1	66	186	1	53	0s03	1	39	0s01	1	27	0s00	18
ii8a2	180	800	1	262	0s01	1	80	0s05	1	62	0s00	30
ii8a3	264	1552	1	465	0s03	1	156	0s08	1	109	0s00	8
ii8a4	396	2798	1	240	0s10	1	287	0s11	1	208	0s01	117
ii8b1	336	2068	1	138	0s05	1	117	0s06	1	105	0s01	18
ii8b2	576	4088	1	11457	1s65	1	437	0s15	1	195	0s01	56
ii8b3	816	6108	1	9406	1s85	1	1530	0s18	1	554	0s03	130
ii8b4	1068	8214	5	17083	24s90	1	2954	0s31	1	450	0s03	14
ii8c1	510	3065	1	277	0s06	1	212	0s11	1	178	0s01	22
ii8c2	950	6689	1	2170	0s46	1	292	0s20	1	269	0s03	19
ii8d1	530	3207	1	505	0s11	1	224	0s05	1	191	0s01	13
ii8d2	930	6547	1	1261	0s33	1	373	0s18	1	355	0s03	9
ii8e1	520	3136	1	334	0s10	1	233	0s11	1	159	0s01	84
ii8e2	870	6121	1	9301	1s40	1	533	0s20	1	256	0s01	9

TAB. 6.6 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances II*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
hanoi4	718	4934		*Non résolu*			*Non résolu*			*Non résolu*		—
hanoi5	1931	14468		*Non résolu*			*Non résolu*			*Non résolu*		—

TAB. 6.7 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances HANOI*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
par16-1-c	317	1264	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-1	1015	3310	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-2-c	349	1392	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-2	1015	3374	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-3-c	334	1332	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-3	1015	3344	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-4-c	324	1292	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-4	1015	3324	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-5-c	341	1360	*Non résolu*			*Non résolu*			*Non résolu*			—
par16-5	1015	3358	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-1-c	1315	5254	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-1	3176	10277	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-2-c	1303	5206	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-2	3176	10253	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-3-c	1325	5294	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-3	3176	10297	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-4-c	1333	5326	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-4	3176	10313	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-5-c	1339	5350	*Non résolu*			*Non résolu*			*Non résolu*			—
par32-5	3176	10325	*Non résolu*			*Non résolu*			*Non résolu*			—
par8-1-c	64	254	2	971	0s10	*Non résolu*			1	49	0s00	10
par8-1	350	1149	*Non résolu*			*Non résolu*			*Non résolu*			—
par8-2-c	68	270	4	992	0s25	*Non résolu*			1	1208	0s01	8
par8-2	350	1157	*Non résolu*			*Non résolu*			*Non résolu*			—
par8-3-c	75	298	*Non résolu*			*Non résolu*			1	95	0s00	7
par8-3	350	1171	*Non résolu*			*Non résolu*			*Non résolu*			—
par8-4-c	67	266	*Non résolu*			*Non résolu*			1	228	0s00	4
par8-4	350	1155	*Non résolu*			*Non résolu*			5	6116	0s38	9
par8-5-c	75	298	*Non résolu*			3	1446	0s10	2	1323	0s03	164
par8-5	350	1171	*Non résolu*			*Non résolu*			*Non résolu*			—

TAB. 6.8 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances PAR*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
g125.17	2125	66272	*Non résolu*			*Non résolu*			*Non résolu*			—
g125.18	2250	70163	*Non résolu*			*Non résolu*			1	7711	1s01	1
g250.15	3750	233965	1	11100	10s60	1	16308	8s65	1	2080	1s71	21
g250.29	7250	454622	*Non résolu*			*Non résolu*			3	133473	110s71	7

TAB. 6.9 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances G*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
aim-100-1_6-yes1-1	100	160		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-1_6-yes1-2	100	160		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-1_6-yes1-3	100	160		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-1_6-yes1-4	100	160		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-2_0-yes1-1	100	200		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-2_0-yes1-2	100	200		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-2_0-yes1-3	100	200		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-2_0-yes1-4	100	200		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-100-3_4-yes1-1	100	340	5	6351	1s76	3	635	0s28	1	1280	0s01	18
aim-100-3_4-yes1-2	100	340	2	2556	0s50	1	6794	0s13	1	132	0s00	9
aim-100-3_4-yes1-3	100	340	1	6075	0s25	3	1066	0s30	1	568	0s00	2
aim-100-3_4-yes1-4	100	340	1	725	0s05	1	2379	0s08	1	686	0s01	13
aim-100-6_0-yes1-1	100	600	1	253	0s03	1	2021	0s10	1	87	0s00	11
aim-100-6_0-yes1-2	100	600	2	1873	0s75	1	1743	0s08	1	98	0s00	11
aim-100-6_0-yes1-3	100	600	1	2729	0s20	1	495	0s06	1	49	0s00	17
aim-100-6_0-yes1-4	100	600	1	636	0s06	1	2509	0s10	1	156	0s00	7
aim-200-1_6-yes1-1	200	320		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-1_6-yes1-2	200	320		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-1_6-yes1-3	200	320		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-1_6-yes1-4	200	320		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-2_0-yes1-1	200	400		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-2_0-yes1-2	200	400		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-2_0-yes1-3	200	400		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-2_0-yes1-4	200	400		*Non résolu*			*Non résolu*			*Non résolu*		—
aim-200-3_4-yes1-1	200	680	1	2730	0s13	2	27906	0s95	1	32301	0s51	21
aim-200-3_4-yes1-2	200	680	1	13330	0s61	3	2171	1s13	1	7959	0s11	13
aim-200-3_4-yes1-3	200	680	2	5681	2s01	3	12356	1s28	1	5783	0s08	16
aim-200-3_4-yes1-4	200	680	1	16741	0s76	2	5895	0s66	1	23267	0s35	21
aim-200-6_0-yes1-1	200	1200	2	5954	3s18	1	17118	0s43	1	154	0s00	10
aim-200-6_0-yes1-2	200	1200	3	1114	5s66	1	2888	0s13	1	267	0s01	14
aim-200-6_0-yes1-3	200	1200	2	35718	5s25	1	1521	0s13	1	174	0s01	48
aim-200-6_0-yes1-4	200	1200	1	436	0s03	1	2253	0s13	1	536	0s01	47
aim-50-1_6-yes1-1	50	80		*Non résolu*			*Non résolu*		1	189	0s00	73
aim-50-1_6-yes1-2	50	80		*Non résolu*			*Non résolu*		2	1033	0s05	128
aim-50-1_6-yes1-3	50	80		*Non résolu*			*Non résolu*		1	660	0s01	96
aim-50-1_6-yes1-4	50	80		*Non résolu*			*Non résolu*		1	407	0s01	65
aim-50-2_0-yes1-1	50	100		*Non résolu*			*Non résolu*		1	174	0s00	17
aim-50-2_0-yes1-2	50	100		*Non résolu*		4	682	0s11	1	954	0s00	13
aim-50-2_0-yes1-3	50	100		*Non résolu*			*Non résolu*		1	474	0s00	96
aim-50-2_0-yes1-4	50	100		*Non résolu*			*Non résolu*		1	374	0s00	9
aim-50-3_4-yes1-1	50	170		*Non résolu*		5	1867	0s20	1	236	0s00	50
aim-50-3_4-yes1-2	50	170	1	403	0s01	1	400	0s01	1	80	0s00	11
aim-50-3_4-yes1-3	50	170	2	597	0s15	2	50	0s10	1	257	0s01	67
aim-50-3_4-yes1-4	50	170	1	616	0s01	1	563	0s06	1	128	0s00	2
aim-50-6_0-yes1-1	50	300	1	203	0s00	1	340	0s06	1	65	0s00	9
aim-50-6_0-yes1-2	50	300	1	338	0s01	1	325	0s03	1	82	0s00	78
aim-50-6_0-yes1-3	50	300	1	495	0s05	1	223	0s05	1	20	0s00	1
aim-50-6_0-yes1-4	50	300	1	442	0s03	1	498	0s05	1	70	0s01	69

TAB. 6.10 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances AIM*

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
alu1	79	196	1	217	0s01	1	70	0s05	1	32	0s00	3
and_or_1	56	86	1	14	0s01	1	43	0s01	1	12	0s00	1
and_or_3	35	53	1	72	0s00	1	104	0s05	1	14	0s00	1
c024	24	164	1	407	0s01	1	19	0s03	1	19	0s01	2
c040	40	340	1	107	0s00	1	160	0s05	1	23	0s00	3
c060	60	610	1	239	0s03	1	3071	0s10	1	59	0s00	16
c084	84	994	1	226	0s01	1	179	0s06	1	49	0s00	13
camp	44	125	1	99	0s01	1	43	0s03	1	22	0s00	7
co14	323	971	1	323	0s06	1	496	0s05	1	158	0s00	21
dc1	79	219	1	90	0s00	1	78	0s01	1	37	0s00	5
example	9	17	1	2	0s00	1	8	0s05	1	3	0s00	1
ftraf	107	293	1	106	0s00	1	81	0s05	1	53	0s00	27
ftraf1	95	256	1	71	0s01	1	37	0s05	1	57	0s00	1
ftraf2	95	256	1	66	0s01	1	98	0s05	1	44	0s01	1
ftraf3	95	256	1	65	0s00	1	59	0s05	1	57	0s00	1
inverter1	30	79	1	191	0s00	1	30	0s03	1	17	0s00	1
nod6col5	75	315	1	522	0s01	1	117	0s05	1	31	0s00	15
random3	170	600	1	4442	0s21	1	587	0s03	1	89	0s00	27
real1a12	18	273	1	2	0s01	1	4	0s05	1	7	0s00	1
real1b12	15	110	1	0	0s01	1	1	0s06	1	1	0s00	1
real1o11	8	20	1	3	0s00	1	2	0s03	1	7	0s00	2
real1q11	16	100	1	4	0s00	1	3	0s05	1	1	0s00	1
real1r11	16	84	1	3	0s00	1	5	0s05	1	3	0s00	1
real1u11	8	24	1	0	0s00	1	3	0s01	1	4	0s00	1
real1v11	8	17	1	1	0s00	1	4	0s05	1	2	0s00	1
real1x11	14	77	1	4	0s00	1	1	0s05	1	3	0s00	1
real1y11	7	32	1	8	0s00	1	1	0s01	1	1	0s00	1
real2a12	18	275	1	10	0s03	1	1	0s03	1	7	0s00	1
real2b12	15	110	1	4	0s00	1	4	0s01	1	1	0s00	1
real2c12	17	342	1	5	0s00	1	3	0s06	1	4	0s01	1
real2d12	17	76	1	5	0s00	1	3	0s05	1	4	0s00	1
real2f12	16	306	1	2	0s01	1	3	0s05	1	1	0s00	1
real2g12	18	123	1	7	0s00	1	3	0s03	1	7	0s00	5
real2j12	17	200	1	1	0s00	1	3	0s05	1	2	0s00	1
real2k12	17	226	1	2	0s00	1	1	0s06	1	1	0s00	1
real2l12	15	119	1	2	0s01	1	3	0s03	1	2	0s00	1
real2m12	11	12	1	2	0s00	1	1	0s05	1	3	0s00	1
real2n12	18	62	1	2	0s00	1	3	0s06	1	4	0s00	1
real2o11	6	18	1	3	0s00	1	2	0s01	1	1	0s00	1
real2o12	15	37	1	2	0s00	1	1	0s05	1	2	0s00	1
real2p11	8	42	1	2	0s01	1	1	0s01	1	1	0s00	1
real2q11	16	100	1	1	0s00	1	3	0s05	1	1	0s00	1
real2r11	16	84	1	4	0s01	1	3	0s06	1	3	0s00	1
real2u11	8	24	1	4	0s00	1	2	0s03	1	4	0s00	1
real2v11	8	17	1	3	0s01	1	4	0s03	1	2	0s00	1
real2w11	7	32	1	2	0s00	1	1	0s03	1	1	0s00	1
real2x11	14	77	1	2	0s01	1	4	0s05	1	3	0s00	1
s040	40	70	1	24	0s00	1	36	0s01	1	25	0s00	4
s060	60	120	1	125	0s00	1	81	0s01	1	35	0s00	20
s084	84	189	1	88	0s00	1	166	0s06	1	39	0s00	11
s112	112	280	1	200	0s01	1	134	0s05	1	55	0s00	33
s144	144	396	1	356	0s01	1	208	0s05	1	57	0s01	30

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
s180	180	540	1	209	0s01	1	326	0s03	1	105	0s00	83
traf	113	313	1	137	0s00	1	81	0s05	1	53	0s00	27
traf2	95	258	1	59	0s00	1	98	0s05	1	44	0s01	1
traf3	107	293	1	133	0s01	1	59	0s05	1	57	0s00	1
ulm027_0	25	64	1	57	0s00	1	19	0s03	1	13	0s00	4
ulm027_1	23	64	1	100	0s01	1	13	0s03	1	44	0s00	5
ulm027_2	25	64	1	19	0s01	1	41	0s06	1	20	0s00	1
ulm054_0	50	128	1	30	0s00	1	26	0s05	1	20	0s00	16
ulm054_1	46	128	1	482	0s01	1	120	0s03	1	86	0s01	54
ulm054_2	50	128	1	89	0s00	1	28	0s06	1	32	0s00	15
ulm081_0	75	192	1	179	0s01	1	150	0s03	1	46	0s00	8
ulm081_1	69	192	1	117	0s01	1	79	0s03	1	41	0s01	15
ulm082_2	75	192	1	133	0s01	1	81	0s06	1	33	0s00	14
ulm108_0	100	256	1	214	0s01	1	94	0s06	1	66	0s00	20
ulm108_1	92	256	1	217	0s01	1	696	0s08	1	113	0s00	5
ulm108_2	100	256	1	135	0s00	1	119	0s01	1	59	0s00	26
ulm135_0	125	320	1	519	0s01	1	280	0s05	1	81	0s00	6
ulm135_1	115	320	1	747	0s01	1	369	0s06	1	91	0s00	2
ulm135_2	125	320	1	158	0s03	1	176	0s03	1	70	0s00	19
ulm162_0	150	384	1	138	0s01	1	144	0s05	1	94	0s00	19
ulm162_1	138	384	1	717	0s03	1	814	0s06	1	491	0s00	135
ulm162_2	150	384	1	226	0s01	1	151	0s05	1	90	0s00	15
ulm189_0	175	448	1	385	0s03	1	174	0s03	1	103	0s00	15
ulm189_1	161	448	1	774	0s03	1	375	0s03	1	317	0s00	3
ulm189_2	175	448	1	149	0s01	1	113	0s05	1	111	0s00	33
ulm216_0	200	512	1	352	0s03	1	225	0s05	1	136	0s00	34
ulm216_1	184	512	1	846	0s05	1	611	0s05	1	325	0s00	50
ulm216_2	200	512	1	508	0s03	1	165	0s05	1	86	0s01	30

TAB. 6.11 – GSAT+RWS vs WSAT+Noise vs TSAT sur les instances FAW

Problèmes	V	C	GSAT+RWS			WSAT+Noise			TSAT			
			#try	#flips	temps	#try	#flips	temps	#try	#flips	temps	lg
jnh1	100	850	1	2261	0s53	1	1015	0s11	1	319	0s01	11
jnh12	100	850	1	782	0s18	1	207	0s08	1	353	0s01	4
jnh17	100	850	1	1087	0s26	1	455	0s10	1	60	0s00	3
jnh201	100	800	1	102	0s06	1	107	0s08	1	98	0s01	2
jnh204	100	800	1	5611	0s91	1	3808	0s16	1	532	0s01	8
jnh205	100	800	1	1933	0s35	1	2018	0s15	1	68	0s01	8
jnh207	100	800	2	2413	2s00	1	717	0s11	1	1811	0s05	4
jnh209	100	800	1	2149	0s36	1	6109	0s21	1	69	0s00	17
jnh210	100	800	1	297	0s06	1	1331	0s10	1	183	0s00	18
jnh212	100	800	*Non résolu*			2	8815	0s58	1	2169	0s06	3
jnh213	100	800	1	674	0s11	1	1620	0s13	1	136	0s01	5
jnh217	100	800	1	876	0s18	1	578	0s05	1	100	0s00	8
jnh218	100	800	1	2782	0s46	1	1984	0s08	1	107	0s01	13
jnh220	100	800	3	1799	3s50	1	9331	0s31	1	887	0s01	2
jnh301	100	900	3	5051	4s88	1	7832	0s31	1	403	0s01	10
jnh7	100	850	1	446	0s10	1	883	0s10	1	67	0s01	10

TAB. 6.12 – *GSAT+RWS vs WSAT+Noise vs TSAT sur les instances JNH*

Le constat réalisable à la lecture de ces tables est que quel que soit le type d'instances (structurées), TSAT semble plus efficace que GSAT+RWS et WSAT+Noise. En effet, pour de nombreuses classes d'instances TSAT résout en moyenne plus d'instances et surtout plus rapidement. Cependant, il nous faut relativiser ce résultat. Le paramétrage de TSAT, en terme de longueur de la liste tabou, est extrêmement délicat. Il ne semble pas exister une longueur privilégiée, même propre à chaque types d'instances. La seule solution est donc de régler manuellement ce paramètre par une recherche ne reposant sur aucun fondement théorique mais basée sur l'intuition. Il nous semble primordial de déterminer une technique permettant d'ajuster automatiquement la longueur de la liste tabou à la longueur optimale quelle que soit l'instance testée.

6.2.5 Vers un réglage dynamique de la liste tabou

L'inconvénient majeur de TSAT réside dans le réglage de la longueur de la liste tabou sur des instances autres qu'aléatoires. P. Battiti et R. Protasi proposent une méthode tabou pour résoudre **MAXSAT** où la longueur de la liste tabou évolue durant la recherche [Battiti & Protasi 1996]. Indépendamment de ces travaux, nous avons testé de multiples approches pour rendre dynamique la gestion de la longueur de la liste tabou.

L'idée de base est de partir d'une liste tabou de longueur 1 (afin d'éviter de reflipper la même variable et de revenir dans l'état précédent) et d'augmenter ou de diminuer (sans jamais descendre en dessous de 1) cette longueur suivant certains critères.

Les premières approches naïves testées consistent à augmenter la liste tabou d'un pas donné dès que l'algorithme commence l'exploration d'un plateau. Sur les instances aléatoires, cette approche aboutit, comparativement au TSAT « standard », à de multiples échecs. Par ailleurs les longueurs finales de la liste tabou étaient plus grandes que les valeurs optimales (cf. paragraphe 6.2.3) observées lors de la mise au point de TSAT. De plus, pour une taille d'instances donnée, l'écart-type entre les différentes valeurs obtenues est relativement conséquent.

Dans ce type d'approche, il est évident qu'un critère permettant de faire diminuer la longueur de la liste tabou est nécessaire. Déterminer un tel critère est délicat. Par exemple, des changements fréquents et brusques peuvent amener à une dégradation des performances.

La réduction de la taille de la liste tabou s'effectue lorsque toutes les variables apparaissant dans des clauses falsifiées appartiennent également à la liste tabou. Intuitivement, il est évident que si toutes les variables susceptibles d'être flippées sont des variables tabou, l'algorithme devient trop contraignant. Il faut donc diminuer la taille de la liste pour sortir de ce cas de figure. Cependant, ce dernier cas est peu fréquent pour permettre une diminution conséquente de la liste tabou. Par conséquent, on obtient les mêmes conclusions avec et sans utilisation de ce critère :

- longueur de liste en fin de recherche trop grande (par rapport à la taille optimale observée) ;
- écart-type important entre les tailles de liste tabou sur des instances aléatoires de même taille ;
- moins performant que TSAT avec un tabou de longueur fixé.

Néanmoins, chacune de ces observations a été atténuée lorsque le critère de diminution est utilisé.

L'inconvénient de ces approches est qu'elles augmentent trop fréquemment la longueur de la liste tabou. Partant de cette conclusion, nous avons essayé de déterminer des paramètres plus pertinents quant à l'augmentation de la longueur de la liste tabou.

L'approche imaginée consiste à rechercher des cycles dans les réparations et d'ajuster la taille de la liste tabou en fonction de la taille des cycles observés. Plus précisément, lorsqu'un minimum local est atteint, chaque variable flippée est mémorisée dans un historique. Lors de chaque nouvelle réparation, l'algorithme parcourt cet historique afin de vérifier si la variable qui s'apprête à être flippée ne provoque pas un cycle dans les réparations (cf. exemple ci-dessous). C'est-à-dire que l'algorithme s'assure que l'on ne revient pas sur une interprétation déjà explorée du minimum local.

Exemple 6.1 (Exemple de cycle lors des réparations)
Supposons que l'approche arrive sur un minimum local (c'est-à-dire que le `score` de la meilleure variable est nul), que la séquence de réparations sur ce plateau est : $\ldots; a; b; c; d; b; a; d$ et que la variable élue pour la prochaine réparation est c.
La sous-séquence $a; b; c; d; b; a; d; c$ est un cycle de taille 4 (4 variables interviennent dans ce cycle).

Si la prochaine variable à flipper entraîne un cycle dans les réparations, la longueur de la liste tabou est fixée à la taille de ce cycle et une nouvelle variable pour la réparation est déterminée.

Il faut noter que cette approche permet de façon évidente d'augmenter la taille de la liste tabou (la taille de la liste tabou détermine la longueur du plus petit cycle détectable). Cependant, dans le même cas de figure que précédemment (toutes les variables susceptibles d'être flippées sont tabou) la taille de la liste peut être réduite. Dans ce dernier cas, des cycles plus petits que la taille actuelle de la liste tabou peuvent survenir. La méthode détecte ces cycles et ajuste, donc diminue, la longueur de la liste tabou.

Cette méthode est certainement la plus prometteuse de toutes les approches testées. En effet, il a été possible de montrer expérimentalement que la taille de la liste tabou obtenue avec cette méthode converge vers la courbe des tailles optimales pour les instances aléatoires. C'est-à-dire qu'après quelques étapes la taille n'évolue plus et se stabilise à la valeur optimale observée lors de la mise au point de TSAT (au-delà, plus aucun cycle dans les réparations n'est détecté).

Par ailleurs, l'efficacité de cette méthode en terme du nombre de problèmes résolus est comparable à celle de TSAT avec une longueur tabou fixe. Cependant le nombre de flips nécessaires à cette méthode est supérieur à celui de TSAT standard. Nous pensons que cet écart du nombre de flips provient du temps passé à faire converger la longueur de la liste tabou. Par ailleurs, cette nouvelle approche est beaucoup plus lente du fait de la recherche des cycles coûteuse en temps (et parfois en mémoire).

L'objectif premier était de réaliser une méthode basée sur TSAT capable d'adapter sa taille de liste tabou pour des problèmes autres qu'aléatoires. Il s'avère qu'aucune de ces méthodes n'a permis réellement d'atteindre cet objectif. En effet, même si la méthode à base de recherche de cycles a permis sur les problèmes aléatoires de stabiliser la taille tabou et de résoudre autant d'instances que TSAT classique, sur les instances structurées les résultats obtenus ne sont pas satisfaisants. Par exemple, sur certaines instances la taille optimale pour TSAT (classique) peut atteindre 80 et il n'a pas été possible pour ces problèmes de détecter des cycles de taille supérieure à 20 ! Il en découle que la plupart du temps, l'algorithme ne trouve pas de modèle.

Cependant, les résultats obtenus avec la méthode à base de cycles sont encourageants et ouvrent de nouvelles voies de recherche dans la mise au point d'un TSAT dynamique.

6.3 Choix d'une interprétation initiale par déséquilibre

Pour de nombreux chercheurs, le problème du choix de l'interprétation initiale est sans importance [Gent & Walsh 1993b] à moins de choisir une interprétation qui soit un modèle ! De plus, la plupart des choix à opérer lors de la résolution d'un problème NP-complet sont coûteux en temps (réaliser le meilleur choix nécessite un surcoût souvent non négligeable), ainsi un choix aléatoire apparaît comme une bonne alternative. Ceci explique que la plus grande partie des algorithmes de recherche locale génèrent l'interprétation initiale de manière aléatoire (*e.g.* GSAT, WSAT, TSAT, etc.).

Cependant, le caractère aléatoire déjà fortement présent dans la plupart des stratégies de réparations (cf. RWS, Noise, etc.) ne facilite pas l'analyse des méthodes de recherche locale. Il est donc intéressant d'atténuer ce caractère aléatoire, à commencer par le choix de l'interprétation initiale.

Les premiers travaux entrepris dans cette voie consistent à fixer les valeurs de vérité de certaines variables v pour lesquelles il est possible d'affirmer que la base de connaissance Σ est consistante si et seulement si Σ_v est consistante. On dit alors que l'on simplifie la base de connaissance. Les méthodes de simplification utilisées sont classiquement la propagation unitaire, l'affectation des littéraux purs, la résolution bornée, etc. [Cha & Iwama 1995, Kask & Dechter 1995]. Les valeurs des variables ne pouvant pas être fixées sont choisies aléatoirement. Il est à noter qu'une telle méthode permet assurément de s'approcher d'un modèle mais également d'imposer les valeurs de vérité de certaines variables qui ne doivent plus être remises en cause lors de la recherche.

D'autres travaux portent sur la génération d'une interprétation initiale à partir des recherches déjà effectuées [Selman & Kautz 1995]. Ces méthodes se rapprochent des méthodes à base d'algorithmes génétiques [Goldberg 1989] et ne s'appliquent que lorsqu'un essai (« try ») a déjà été effectué. Une première idée consiste à choisir comme nouvelle interprétation de départ, la meilleure interprétation (c'est-à-dire celle qui minimise le plus la fonction d'évaluation) du « try » précédent en lui faisant subir quelques mutations. Une mutation consiste à choisir un certain nombre de variables au hasard et à inverser leur valeur de vérité. Une seconde approche consiste à construire à partir des deux meilleures interprétations du « try » précédent (ou des deux « tries » précédents) une interprétation telle que les littéraux communs aux deux interprétations soient conservés et les autres soient choisis aléatoirement.

Algorithme 6.13 INTERPRÉTATIONS PAR DÉSÉQUILIBRE

```
1.  Function INIT_INTERPRÉTATION_PAR_DÉSÉQUILIBRE :
2.  Input  : Un ensemble de clauses Σ, un entier try correspondant au try actuel ;
3.  Output : Une interprétation initialisée suivant les valeurs de déséquilibre ;
4.  Begin
5.      I=∅ ;                                          %% Interprétation retournée
6.      foreach variable v ∈ Σ do
7.          Calculer des[v] le déséquilibre de la variable v ;
8.          if (abs(des[v]) > (try-1)) then
9.             if (des[v] > 0) then I=I ∪ {v} else I=I ∪ {¬v} fi
10.         else I=I ∪ {x}, avec au hasard x = v ou x = ¬v ;
11.         fi
12.     done
13.     return I ;
14. End
```

Nous proposons une manière de choisir l'interprétation initiale en fonction du déséquilibre des variables. Nous rappelons que le déséquilibre d'une variable est la différence entre son nombre d'occurrences positives et son nombre d'occurrences négatives. La configuration initiale est donc calculée par l'intermédiaire de cette notion, c'est-à-dire qu'une variable propositionnelle ayant un déséquilibre positif (négatif) sera affectée à $\mathbb{V}$ (respectivement à $\mathbb{F}$). Si une variable possède un déséquilibre nul (autant d'occurrences positives que négatives), alors la valeur de vérité de cette variable est choisie aléatoirement.

Par ailleurs, afin que les interprétations initiales de chaque « try » ne soient pas trop proches les unes des autres, plus le nombre de « tries » est élevé, plus la proportion de valeurs déterminées par l'intermédiaire du déséquilibre diminue. En effet, au cours du i*ème* essai, seules les variables possédant un déséquilibre (en valeur absolue) strictement supérieur à $i - 1$ sont affectées en fonction du déséquilibre. La fonction générant de telles interprétations initiales est détaillée page précédente.

Choisir l'interprétation initiale permet évidemment de diminuer le nombre de clauses falsifiées et de fixer correctement la valeur des littéraux impliqués. En effet, plus le déséquilibre d'une variable est grand, plus cette variable a de fortes chances d'être impliquée.

Cette notion de déséquilibre a été introduite dans le but de fixer un maximum de variables à leur valeur obligatoire. C'est le cas, par exemple des instances *k***SAT** aléatoires satisfaisables au seuil, ou de nombreux problèmes à solution unique. Néanmoins, contrairement aux simplifications présentées précédemment, ces valeurs peuvent être remises en cause lors de la recherche locale.

Génération initiale testée		*Type de données*	*Moyenne*	*Valeur minimale*	*Valeur maximale*	*Ecart Type*
Aléatoire		Distance Moyenne	88,30	68,84	96,34	3,99
		Valeur minimale	69,02	51	79	3,70
		Valeur maximale	106,75	84	118	6,15
		Ecart type	6,27	5,09	6,15	0,53
Déséquilibre	Try 1	Distance Moyenne	57,88	51,27	64,02	2,20
		Valeur minimale	43,43	37	50	1,71
		Valeur maximale	78,17	72	84	2,76
		Ecart type	6,38	6,36	6,77	0,24
	Try 2	Distance Moyenne	60,20	49,16	71,12	3,66
		Valeur minimale	46,88	35	59	2,96
		Valeur maximale	78,23	68	89	4,38
		Ecart type	5,98	6,01	6,88	0,32
	Try 3	Distance Moyenne	64,51	50,86	77,93	4,53
		Valeur minimale	53,41	40	63	3,60
		Valeur maximale	80,14	69	96	5,29
		Ecart type	5,41	5,29	6,77	0,40
	Try 4	Distance Moyenne	70,07	54,42	85,14	5,11
		Valeur minimale	60,08	44	77	3,97
		Valeur maximale	83,64	67	101	6,15
		Ecart type	5,13	4,61	6,60	0,49
	Try 5	Distance Moyenne	75,18	58,32	91,83	5,55
		Valeur minimale	64,71	50	78	4,47
		Valeur maximale	85,98	69	110	6,39
		Ecart type	4,50	4,17	6,69	0,49

TAB. 6.13 – *Interprétations « aléatoires » et « déséquilibrées » : distances au plus proche modèle*

Afin de valider expérimentalement une telle méthode de génération d'interprétations initiales, nous avons effectué les comparaisons suivantes. Ces expérimentations ont porté sur une centaine d'instances *3***SAT** au seuil ($C/V = 4,25$) à 200 variables. Pour chacune de ces instances, l'ensemble de ses modèles a été calculé, ceci afin de mesurer la distance minimale séparant un modèle d'une interprétation initiale. Par ailleurs, pour chaque type de générateur d'interprétations initiales (initialisation aléatoire ou initialisation avec déséquilibre pour les 5 premiers « tries »), nous avons effectué 500 tirages d'interprétations initiales. A chaque tirage, la distance minimale en terme de HAMMING, entre l'interprétation initiale et l'ensemble des modèles a été mesurée. Pour ces 500 plus petites distances, les valeurs minimales et maximales ont été mémorisées et la moyenne ainsi que l'écart-type ont été calculés. La table 6.13 page précédente synthétise pour l'ensemble des problèmes testés et pour chaque type de génération, les minima, maxima, moyennes et écarts types des valeurs précédemment décrites.

À la vue de cette table, il apparaît que la méthode de génération par déséquilibre permet de manière très nette d'obtenir des interprétations initiales plus proches d'un modèle que le générateur d'interprétations aléatoires (cf. lignes « Distance Moyenne » et colonne « Moyenne » de la table 6.13) et cela quel que soit le « try » considéré. Par ailleurs, plus le nombre de variables fixées grâce au déséquilibre est grand plus la distance à un modèle diminue. Cette dernière constatation montre toute l'importance des littéraux déséquilibrés.

Cependant, obtenir une interprétation initiale proche d'un modèle n'est pas forcément synonyme de trouver un modèle plus rapidement. Afin d'infirmer ou de valider expérimentalement cette hypothèse, nous avons réalisé une étude expérimentale des deux algorithmes GSAT+RWS et TSAT, avec et sans utilisation du déséquilibre.

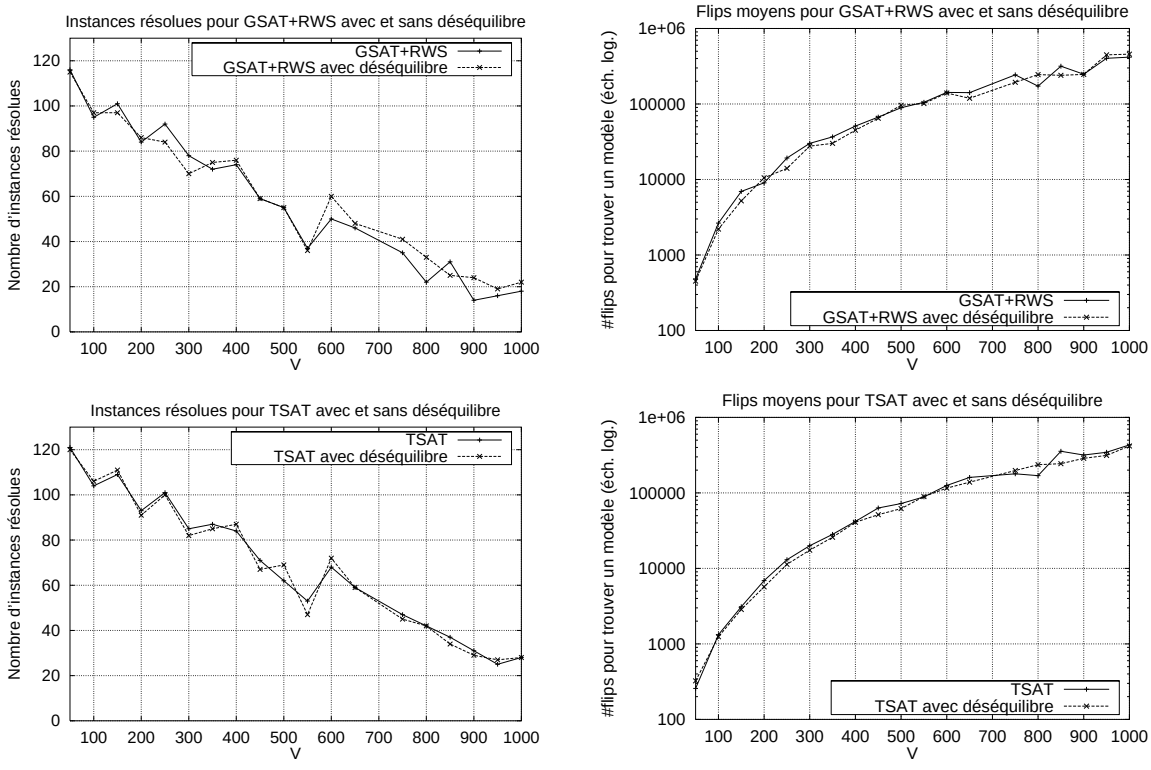

FIG. 6.4 – *Effet d'une interprétation initiale par déséquilbre sur GSAT+RWS et TSAT pour 3***SAT**

Les premières séries de tests ont porté sur des instances aléatoires *3***SAT** au seuil ($C/V = 4,25$) pour un nombre de variables variant de 50 à 1000 par pas de 50 ; 200 instances de chaque taille ont été testées. Les ressources pour chacun des quatre algorithmes sont identiques et ont été imposées à 1 « try » et V^2 « flips ». Par ailleurs la probabilité pour « Random Walk Strategy » a été fixée à 50 et la longueur de la liste tabou a été calculée par l'intermédiaire de la fonction affine proposée précédemment (cf. paragraphe 6.2.3). Les courbes (cf. FIG. 6.4) représentent respectivement pour les quatre algorithmes le nombre de problèmes résolus et le nombre moyen de flips nécessaires pour exhiber un modèle.

Les courbes avec et sans déséquilibre sont très proches pour les algorithmes testés. Par conséquent, pour GSAT+RWS et TSAT, générer l'interprétation initiale au moyen du déséquilibre des variables ne permet pas d'obtenir une nette amélioration des performances sur des instances *3***SAT** aléatoires. Notre intuition est qu'il en est de même pour toutes les méthodes de recherche locale développées jusqu'à présent. Aucune de ces méthodes ne semble être capable d'exploiter le fait que l'interprétation initiale soit sensiblement plus proche d'un modèle qu'à l'accoutumée.

Les mêmes conclusions peuvent être avancées pour des instances structurées. En effet, nous avons testé l'influence d'une interprétation initiale générée grâce au déséquilibre sur une grande partie des instances structurées proposées au challenge DIMACS [DIM1993]. La table 6.14 page suivante présente les résultats obtenus sur les instances[8] « *ii* » et « *par* » pour les algorithmes GSAT+RWS et TSAT avec et sans déséquilibre. L'apport de l'interprétation initiale générée par le déséquilibre des variables est relativement faible et même parfois néfaste pour ces instances, comme pour toutes les autres instances structurées de DIMACS.

Les résultats obtenus sur l'interprétation initiale générée par l'intermédiaire des variables déséquilibrées tendent à atténuer l'affirmation courante minimisant l'importance de l'interprétation initiale. En effet, nous avons montré expérimentalement que générer une interprétation au moyen du déséquilibre permet d'obtenir des interprétations sensiblement plus proches des modèles. Par ailleurs, l'influence de ce type de génération sur les performances des méthodes à recherche locale reste encore à prouver. Cependant, il nous semble, à notre connaissance qu'aucun algorithme de recherche locale n'exploite le déséquilibre des variables dans sa stratégie de réparation. Nous pensons que la mise au point d'une telle stratégie de recherche locale permettrait d'exploiter au mieux la génération d'interprétations initiales au moyen du déséquilibre des variables.

6.4 Conclusions

L'efficacité pratique des méthodes de recherche locale pour la résolution du problème **SAT** n'a été constatée que récemment. Depuis, les travaux sur ces méthodes appliquées à **SAT** foisonnent dans la littérature et il ne fait, d'autre part, aucun doute que la recherche locale a fortement contribué au regain d'intérêt sur **SAT** constaté ces dernières années. De plus, ces méthodes ont permis d'énormes progrès dans la résolution pratique de **SAT** (*e.g.* résolution d'instances consistantes dont la taille atteint plus 10000 variables).

Notre contribution à ces méthodes de recherche locale est multiple :

– mise au point d'une méthode incomplète de réparation locale à base de tabou aux performances

8. Une description des instances proposées à [DIM1993] est fournie en annexe.

Algorithmes			GSAT+RWS	GSAT+RWS avec des.	TSAT	TSAT avec des.
Instances	V	C	Nombre de « flips »			
ii16a1	1650	19368	1363	672	740	449
ii16a2	1602	23281	22463	*Non résolu*	7529	2187
ii16b1	1728	24792	6662	5931	*Non résolu*	*Non résolu*
ii16b2	1076	16121	18899	*Non résolu*	*Non résolu*	*Non résolu*
ii16c1	1580	16467	1739	2461	654	1085
ii16c2	924	13803	49894	1760	*Non résolu*	*Non résolu*
ii16d1	1230	15901	5008	2255	5415	1644
ii16d2	836	12461	33313	4328	*Non résolu*	*Non résolu*
ii16e1	1245	14766	1545	3077	591	917
ii16e2	532	7825	148335	2053	354	*Non résolu*
ii32a1	459	9212	1490	2125	4003	4741
ii32b1	228	1374	815	650	145	253
ii32b2	261	2558	1511	878	1862	336
ii32b3	348	5734	1948	1654	273	390
ii32b4	381	9618	5214	4168	257	1500
ii32c1	225	1280	619	451	101	209
ii32c2	249	2182	436	1384	121	268
ii32c3	279	3272	1360	856	471	576
ii32c4	759	20862	9334	4081	*Non résolu*	*Non résolu*
ii32d1	332	2703	1102	1789	1331	2334
ii32d2	404	5153	2736	1771	7296	*Non résolu*
ii32d3	824	19478	14605	10960	*Non résolu*	*Non résolu*
ii32e1	222	1186	261	341	90	188
ii32e2	267	2746	503	2065	202	275
ii32e3	330	5020	1327	1840	609	402
ii32e4	387	7106	1552	1590	205	323
ii32e5	522	11636	3344	2818	393	746
ii8a1	66	186	126	9	45	11
ii8a2	180	800	164	28	58	27
ii8a3	264	1552	190	150	106	50
ii8a4	396	2798	792	279	207	133
ii8b1	336	2068	153	142	71	60
ii8b2	576	4088	1385	607	310	196
ii8b3	816	6108	4873	8886	411	273
ii8b4	1068	8214	8781	2480	512	451
ii8c1	510	3065	274	104	126	61
ii8c2	950	6689	1622	331	254	165
ii8d1	530	3207	368	172	187	64
ii8d2	930	6547	1921	658	248	148
ii8e1	520	3136	314	426	170	79
ii8e2	870	6121	991	213	228	144
par8-1-c	64	254	*Non résolu*	*Non résolu*	49	115
par8-1	350	1149	*Non résolu*	*Non résolu*	81408	38025
par8-2-c	68	270	4594	*Non résolu*	1208	251
par8-2	350	1157	*Non résolu*	*Non résolu*	29349	21369
par8-3-c	75	298	*Non résolu*	*Non résolu*	95	816
par8-3	350	1171	*Non résolu*	*Non résolu*	*Non résolu*	*Non résolu*
par8-4-c	67	266	*Non résolu*	*Non résolu*	228	1753
par8-4	350	1155	*Non résolu*	*Non résolu*	79622	28679
par8-5-c	75	298	*Non résolu*	*Non résolu*	4560	1633
par8-5	350	1171	*Non résolu*	*Non résolu*	82923	*Non résolu*

TAB. 6.14 – *GSAT+RWS et TSAT : interpétation initiale par déséquilbre sur « ii » et « par-8 »*

comparables aux meilleures approches connues ;
- mise en évidence de phénomènes remarquables quant aux réglages de cette méthode : linéarité de la courbe des longueurs optimales de la liste tabou, laquelle ne dépend uniquement que du nombre de variables pour les instances *k***SAT** ;
- mise au point d'un TSAT auto-adaptatif à base de recherche de cycles dans les réparations ;
- introduction d'une nouvelle méthode de génération de l'interprétation initiale permettant d'obtenir une interprétation plus proche d'un modèle en terme de la distance de HAMMING.

Par ailleurs, les travaux réalisés ouvrent de nouvelles pistes que nous souhaitons explorer.

TSAT est un algorithme dont l'ossature est proche de celle de GSAT. Cependant à notre connaissance, aucune approche à base de tabou **systématique** n'a été développée pour l'algorithme WSAT. Il serait intéressant de mettre au point une telle approche et d'en étudier la courbe des longueurs tabou optimales ainsi que son efficacité.

Par ailleurs, une interprétation initiale générée à partir du déséquilibre des variables permet d'obtenir une interprétation plus proche d'un modèle que l'interprétation aléatoire couramment utilisée. Le problème est qu'à l'heure actuelle aucune stratégie de réparations locales n'exploite cette information afin d'accroître son efficacité. Il serait intéressant de savoir dans quelle mesure il est possible de réaliser un algorithme de recherche locale exploitant cette constatation et de le comparer aux approches classiques.

Chapitre 7

L'algorithme de Davis & Putnam

Hormis les différentes techniques complètes utilisant le principe de résolution (cf. paragraphe 5.3.2.1), la plupart des algorithmes complets pour **SAT** ne sont que des variantes et des améliorations de la procédure bien connue de Davis & Putnam (DP). Néanmoins d'autres techniques complètes existent pour **SAT** (*e.g.* H2R, basée sur un hyper-graphe orienté [Pretolani 1993]), cependant aucune ne rivalise en efficacité avec les algorithmes basés sur DP.

La méthode présentée dans [Davis & Putnam 1960] est un algorithme de résolution. Il contient des éléments de base d'une procédure énumérative présentée dans [Davis *et al.* 1962]. Communément (mais abusivement) la procédure proposée par M. Davis, G. Logemann et D. Loveland se trouve dans la littérature sous le nom de DP ! C'est cette dernière procédure qui fait l'objet de notre étude dans ce chapitre. Les apparitions ultérieures de la procédure de Davis & Putnam ou de l'abréviation DP font référence à l'algorithme proposé dans [Davis *et al.* 1962].

La première partie de ce chapitre est consacrée à la présentation de DP, de différentes propriétés utilisées en vue de son amélioration, ainsi que les heuristiques de branchement les plus fréquemment utilisées. La seconde partie traite de notre contribution à l'amélioration des performances de DP. Plus précisément nous proposons une généralisation du théorème de partition du modèle énoncé initalement dans [Jeannicot *et al.* 1988]. L'étude réalisée dans ce chapitre porte essentiellement sur des techniques de simplification introduites dans DP pour élaguer l'arbre de recherche. Nous proposons dans le chapitre 8 différents schémas de coopération entre algorithmes complets (DP) et incomplets (recherche locale).

7.1 Présentation

DP est l'algorithme énumératif de référence pour la résolution pratique de **SAT** (*e.g.* [Hooker 1993], [Oxusoff & Rauzy 1989, Billionnet & Sutter 1992, Crawford & Auton 1993, Dubois *et al.* 1996, Li 1996, Boufkhad 1996], etc.). Il effectue une énumération systématique de l'ensemble des interprétations afin de déterminer si l'une d'entre elles est un modèle. Par sa simplicité elle est facile à mettre en œuvre et est généralement efficace.

De nombreuses présentations de la procédure DP ont déjà été réalisées. Nous ne nous hasardons donc pas à en faire une description des plus originales. Au contraire, la présentation faite page suivante se veut la

plus simple et la plus élémentaire possible.

Algorithme 7.1 DP

```
1.  Function DP : boolean
2.  Input  : un ensemble Σ de clauses
3.  Output : true si Σ est consistant, false sinon
4.  Begin
5.     Σ*=Propagation_Unitaire(Σ) ;                          %% cf. algorithme 7.2
6.     Σ+=Simplification_Littéraux_Purs(Σ*) ;                %% cf. algorithme 7.3
7.     if (Σ* contient la clause vide) then return false ;
8.     elif (Σ+==∅) then return true ;
9.     else
10.      l=Heuristique_de_branchement(Σ+) ;
11.      return (DP(Σ+ ∪ {(l)})) || (DP(Σ+ ∪ {(¬ l)})) ;
                    %% (l) (resp. (¬ l)) représente la clause unitaire {l} (resp. {¬ l})
12.    fi
13. End
```

Les fonctions `Progagation_Unitaire` et `Simplification_Littéraux_Purs` consistent à supprimer de la base de connaissances respectivement toutes les clauses unitaires et toutes les clauses contenant un littéral pur. Les littéraux complémentaires à un littéral unitaire sont également effacés de la base de connaissances. Les fonctions de simplification jouent un rôle important dans la procédure DP, nous présentons dans le paragraphe 7.1.1 ces procédures ainsi que d'autres techniques de simplification.

Cependant, la fonction critique pour l'efficacité de DP est l'heuristique de branchement (`Heuristique_de_branchement`). Le choix du prochain littéral à affecter influe considérablement sur la taille de l'arbre de recherche, et par conséquent sur les temps d'exécution. Les stratégies de branchement ont fait l'objet de nombreuses études (*e.g.* [Jeroslow & Wang 1990, Billionnet & Sutter 1992, Dubois *et al.* 1996], etc.), nous décrivons dans la section 7.1.3 les heuristiques les plus fréquement employées dans la littérature.

La procédure DP construit implicitement un arbre binaire de recherche dont :
- la racine est l'ensemble de clauses initial ;
- les branches sont étiquetées par des littéraux ;
- les nœuds sont des ensembles de clauses ; ces ensembles de clauses sont obtenus en simplifiant l'ensemble de clauses, du nœud précédent, par le littéral qui étiquette la branche reliant les deux nœuds ;
- les feuilles sont soit l'ensemble vide de clauses (succès), soit un ensemble de clauses contenant la clause vide (échec).

La base de connaissance est prouvée inconsistante lorsque l'arbre de recherche ne contient aucune feuille représentant l'ensemble vide de clauses. L'exemple 7.1 illustre nos propos et montre l'arbre de recherche de DP sur deux instances (une consistante et une inconsistante).

Exemple 7.1 (Arbres de recherche de DP)
Considérons l'ensemble de clauses $\Sigma = \{(a \vee b \vee \neg c), (\neg a \vee c \vee \neg b), (\neg c \vee b \vee \neg a \vee d), (\neg c \vee d), (\neg c \vee \neg d)\}$.

L'arbre de recherche construit par DP en utilisant l'ordre lexicographique comme heuristique de branchement est donné par la figure 7.1. Cet arbre admet une feuille contenant l'ensemble vide de clauses, correspondant au modèle $\{a, \neg b, \neg c\}$ de Σ.

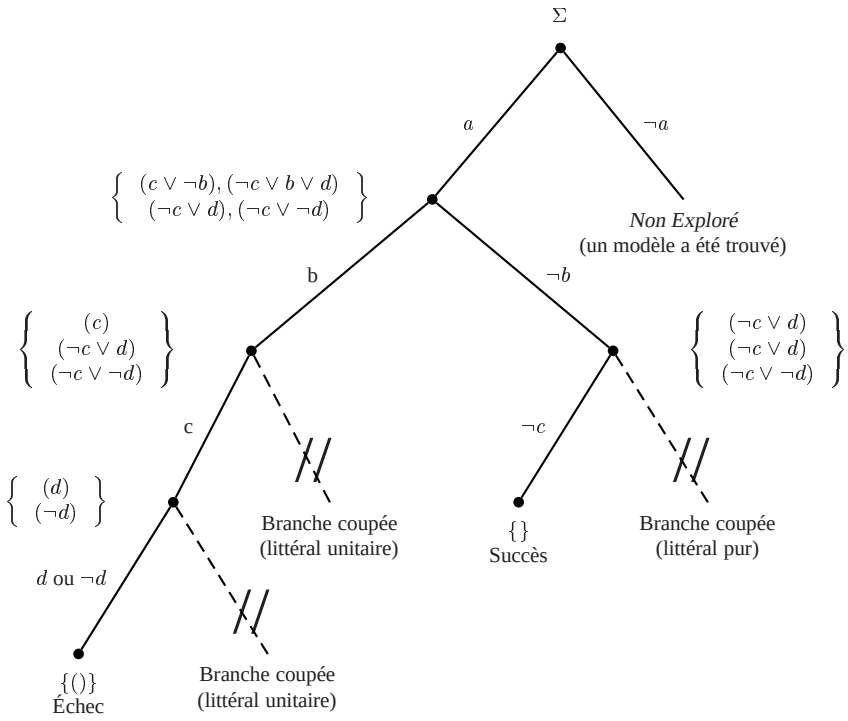

FIG. 7.1 – *Arbre de recherche constuit par Davis & Putnam sur un instance satisfaisable*

Soit $\Omega = \Sigma \cup \{(\neg a \vee b), (a \vee c \vee \neg b), (b \vee c)\}$.

L'arbre décrit par DP pour l'ensemble de clauses Ω est représenté à la figure 7.2 page suivante. Cet arbre n'admettant pas de feuille représentant l'ensemble vide de clauses, Ω est inconsistant.

7.1.1 Méthodes de simplification

Les fonctions `Progagation_Unitaire` et `Simplification_Littéraux_Purs` retournent la base de clauses « *simplifiée* ». Elles s'inscrivent dans un cadre plus général de mécanismes de simplification. L'objectif de ces simplifactions est d'améliorer l'efficacité de l'algorithme en :

- réduisant la taille de la base de clauses ;
- élaguant l'arbre de recherche ;
- améliorant l'efficacité de l'heuristique de branchement.

Cependant, afin que l'algorithme DP conserve sa complétude logique, ces fonctions doivent respecter la règle suivante :

$\Sigma_{\text{simplifiée}}$ *est consistant si et seulement si* Σ *est consistant.*

Par ailleurs, plus la simplification est judicieuse et significative, plus la taille de l'arbre de recherche construit par DP est réduite. En contrepartie, bien souvent, plus la simplification est complexe, plus les temps de calcul à chaque nœud sont importants. Une fonction de simplification doit donc réaliser un compromis entre temps de calcul et réduction du nombre de nœuds. De plus, une méthode de simplification est particulièrement intéressante, si elle permet non seulement de réduire la taille de l'arbre mais également le

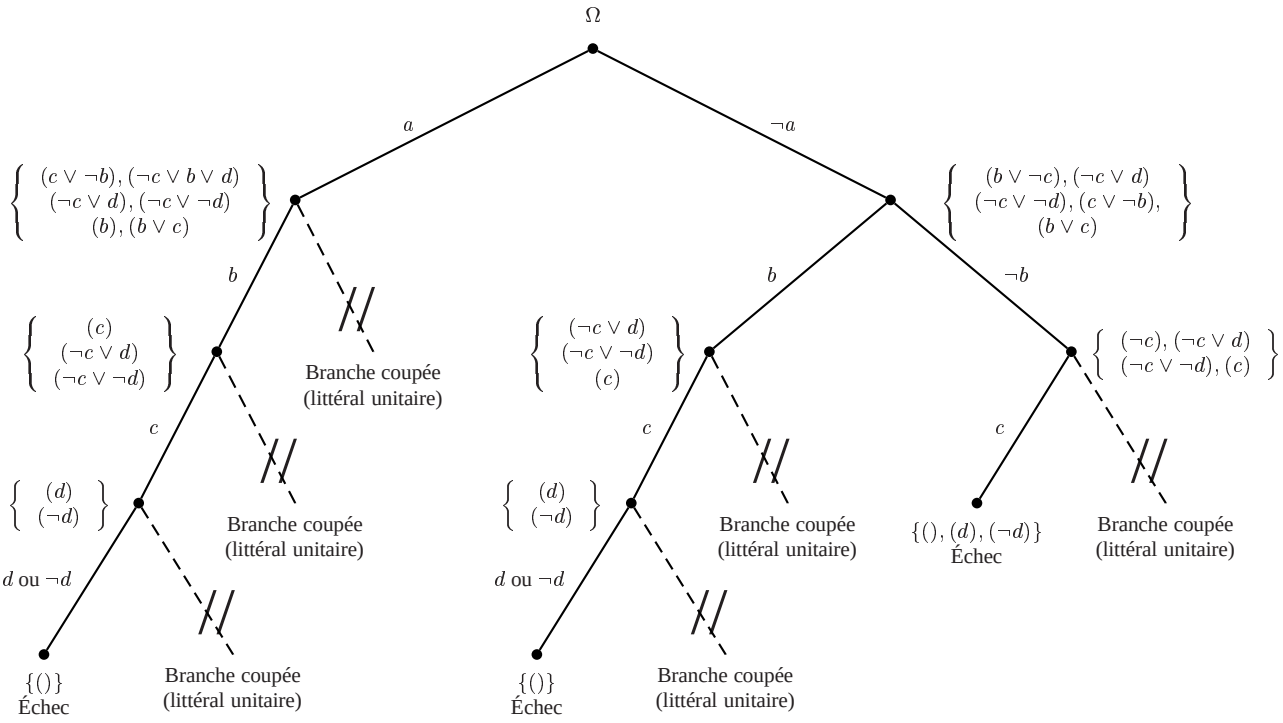

FIG. 7.2 – *Arbre de recherche construit par Davis & Putnam sur une instance insatisfaisable*

temps de calcul. Le coût de la simplification est par conséquent un paramètre non négligeable qui conduit souvent à déterminer les conditions de son utilisation pour l'obtention d'un gain optimal.

Nous présentons une liste non exhaustive des méthodes de simplification les plus fréquemment utilisées.

7.1.1.1 Propagation unitaire

La propagation unitaire repose sur une propriété élémentaire :

Propriété 7.1
Soit Σ un ensemble de clauses. Si l est un littéral unitaire de Σ, alors :

$$\Sigma \text{ est satisfaisable si et seulement si } \Sigma_l \text{ est satisfaisable.}$$

La simplification par un littéral unitaire consiste à supprimer de la base de clauses, toutes les clauses contenant le littéral unitaire et à supprimer toutes les occurrences du littéral complémentaire, c'est-à-dire « raccourcir » les clauses contenant le littéral complémentaire. La propagation unitaire est l'application répétée 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. Nous présentons (cf. algorithme 7.2) une version récursive de la propagation unitaire.

De la propriété 7.1 il découle naturellement que :

Propriété 7.2
Un ensemble de clauses Σ est consistant si et seulement si Σ^* est consistant.

Nous rappelons que la notation Σ^* représente la base de clauses simplifiée par la propagation unitaire (cf. définition 2.36).

Cette simplification est fondamentale pour l'algorithme DP. Généralement, la majeure partie du temps d'exécution est consacrée à la propagation des littéraux unitaires, comme nous le montrons dans le chapitre 9. En outre, la propagation unitaire entraîne une diminution du nombre de clauses, du nombre de variables et de la longueur de certaines clauses. Cette opération permet donc une simplification extrêmement significative de la base de clauses, qui réduit assurément l'arbre de recherche de DP.

Algorithme 7.2 PROPAGATION UNITAIRE

```
1.  Function Propagation_Unitaire : un ensemble de clauses
2.  Input  : Σ un ensemble de clauses ;
3.  Output : Σ* l'ensemble de clauses Σ simplifié par la propagation unitaire
4.  Begin
5.    if (Σ contient la clause vide) then return Σ ;
6.    elif (Σ contient une clause unitaire c = {l})
7.    then
8.         Supprimer de Σ toutes les clauses contenant l ;
9.         Supprimer dans les clauses de Σ toutes les occurrences de ∼l ;
10.        return Propagation_Unitaire(Σ) ;
11.   else return Σ ;
12.   fi
13. End
```

La propagation unitaire se retrouve dans toutes les implantations de DP, elle représente d'ailleurs le dénominateur commun des procédures DP présentées dans [Davis & Putnam 1960] et dans [Davis *et al.* 1962]. En effet dans [Davis & Putnam 1960], la propagation unitaire consiste en une application prioritaire de la règle de résolution sur les clauses unitaires suivie d'une étape de sous-sommation. Par ailleurs, la propagation unitaire requiert un temps d'exécution linéaire [Dowling & Gallier 1984]. Nous rappelons que cette opération est complète pour le test de satisfaisabilité d'un ensemble de clauses de Horn.

7.1.1.2 Simplication par les littéraux purs

La simplification par les littéraux purs repose sur la propriété suivante :

Propriété 7.3
Soit Σ un ensemble de clauses. Si l est un littéral pur de Σ, alors :

Σ est satisfaisable si et seulement si Σ_l est satisfaisable.

Par conséquent, la simplification d'une base de clauses par un littéral pur consiste à supprimer de la base toutes les clauses contenant ce littéral pur. Nous notons $\Sigma^\diamond$ la base de clauses Σ simplifiée par l'ensemble de tous les littéraux purs. Σ et $\Sigma^\diamond$ sont équivalentes du point de vue de la satisfaisabilité :

Propriété 7.4
Σ est consistant si et seulement si $\Sigma^\diamond$ est consistant.

Comme pour la propagation unitaire, la simplification par les littéraux purs est peu coûteuse et peut permettre un gain de temps non négligeable. On retrouve cette simplification dans la plupart des implantations de DP. Nous discutons dans le chapitre 9 son taux d'application en fonction du rapport nombre de clauses sur nombre de variables (C/V) pour les instances *3***SAT** aléatoires.

Nous proposons une version récursive de la procédure de simplification par les littéraux purs. Par ailleurs, nous rappelons que nous notons Σ^+ l'ensemble de clauses simplifié par les littéraux unitaires et purs (cf. définition 2.36) : $\Sigma^+ = (\Sigma^*)^\diamond$.

Algorithme 7.3 SIMPLIFICATION LITTÉRAUX PURS

```
1.  Function Simplification_Littéraux_Purs : un ensemble de clauses
2.  Input  : Σ un ensemble de clauses ;
3.  Output : Σ◇ l'ensemble de clauses Σ simplifié par les littéraux purs
4.  Begin
5.    if (Σ contient un littéral pur l)
6.    then
7.         Supprimer de Σ toutes les clauses contenant l ;
8.         return Simplification_Littéraux_Purs(Σ) ;
9.    else return Σ ;
10.   fi
11. End
```

7.1.1.3 Suppression des clauses sous-sommées

Si dans un ensemble de clauses Σ, une clause c_1 est sous-sommée par une clause c_2 (cf. définition 2.30), nous savons que c_1 est une conséquence logique directe de c_2 : $c_2 \models c_1$. Par conséquent, il est clair que Σ et l'ensemble de clauses Σ simplifié par la suppression des clauses sous-sommées sont équivalents du point de vue de la satisfaisabilité.

Cependant, cette opération est extrêmement coûteuse en temps, elle ralentit donc considérablement l'exécution de DP. Par conséquent, cette opération n'est en général pas réalisée car elle coûte souvent beaucoup plus cher qu'elle ne rapporte, on se contente très fréquemment de supprimer les clauses sous-sommées uniquement à la racine de l'arbre de recherche de DP.

7.1.1.4 Résolution restreinte

La résolution restreinte consiste à ajouter à la base un certain nombre de clauses produites par résolution. Il faut bien entendu limiter cette étape afin de ne pas transformer DP en un algorithme de preuve par résolution (ce dernier étant dans la plupart des cas beaucoup moins efficace que DP). Diverses restrictions peuvent être mises en place. Ces restrictions sont plus ou moins complexes et plus ou moins bénéfiques pour DP (voir par exemple [Génisson & Siegel 1994, Castell 1996, Van Gelder & Tsuji 1996], etc.).

La forme de résolution restreinte la plus fréquemment employée est certainement la résolution bornée. La résolution bornée consiste à ne produire que les résolvantes qui sont de longueur inférieure ou égale à une constante fixée (la borne).

Dans [Billionnet & Sutter 1992], il est proposé d'ajouter la résolvante de deux clauses de la base, si cette résolvante est de longueur inférieure ou égale à k et si cette résolvante n'est pas sous-sommée par une clause de la base.

Dans [Boufkhad 1996], une résolvante entre deux clauses est prise en compte, si la taille de la clause produite est inférieure ou égale au minimum des longueurs des deux clauses considérées. Cette méthode est utilisée dans C-SAT comme pré-traitement [Dubois *et al.* 1996]. Ce dernier permet de détecter certaines inconsistances locales (cf. définition 8.2). Nous proposons dans le chapitre 8, une technique à base de recherche locale permettant également de détecter et de circonscrire des noyaux inconsistants.

Remarque 7.1
Il est à noter que contrairement aux trois premières méthodes de simplification présentées, la résolution restreinte n'est pas synonyme de réduction syntaxique, c'est-à-dire qu'aucune suppression de clauses ou de littéraux n'est effectuée. Au contraire, la résolution bornée procède par ajouts de contraintes (de clauses), ceci afin de simplifier (réduire) l'arbre de recherche décrit par DP.

7.1.1.5 Symétrie

La notion de symétrie (cf. paragraphe 5.1) [Krishnamurthy 1982, Krishnamurthy 1985] a été introduite dans la procédure de Davis & Putnam dans [Benhamou & Saïs 1992, Saïs 1993]. Les symétries permettent parfois une réduction importante de l'espace de recherche. En effet nous avons :

Propriété 7.5 ([Benhamou & Saïs 1994])
Si $l_1, l_2, \ldots, l_n$ un ensemble de littéraux symétriques (un cycle de symétrie) d'un ensemble de clauses Σ, alors

$$\Sigma \text{ est consistant si et seulement si } \begin{cases} \Sigma_{l_1} \text{ est consistant} \\ \text{ou} \\ \Sigma_{\{\sim l_1, \sim l_2, \ldots, \sim l_n\}} \text{ est consistant.} \end{cases}$$

Pour un ensemble de clauses Σ contenant V variables, il y a 2^V interprétations potentielles à explorer. Si Σ contient S littéraux symétriques, le nombre d'interprétations à explorer est réduit à $2^{V-1} + 2^{V-S}$.

Cependant le problème de la détection des symétries est un problème difficile (polynomialement équivalent à l'isomorphisme de graphes, problème ISO-complet, cf. page 45). Il faut souvent se contenter d'un algorithme incomplet pour rechercher les symétries. Les substitutions trouvées sont souvent suffisantes pour permettre à DP une résolution très efficace de problèmes à fortes régularités (structurés) tels que le

problème des « tiroirs et chaussettes » ou encore des problèmes de coloriage de graphes comme les problèmes de « *Ramsey* » [Saïs 1993, Benhamou & Saïs 1994]

7.1.1.6 Simplification par littéraux impliqués

Si dans un ensemble de clauses Σ nous savons qu'un littéral (non unitaire) l est impliqué par Σ, alors Σ est consistant si et seulement si Σ_l [1] est consistant. De cette propriété, plusieurs auteurs ont développé des techniques permettant de simplifier l'arbre de recherche de DP par la mise en évidence de littéraux impliqués.

Dubois *et al.* ([Dubois *et al.* 1996, Boufkhad 1996]) proposent de propager un à un les littéraux apparaissant dans les clauses binaires. Cette opération est réalisée entre deux profondeurs (fixées empiriquement) de l'arbre de DP et l'ordre de traitement des littéraux s'effectue suivant la fonction d'évaluation du littéral (cf. $Poids_{lit}$ section 7.1.3.2). Si un littéral l est impliqué par la propagation unitaire, la clause unitaire $\{l\}$ est ajoutée à la base.

Cette opération permet de simplifier de manière non négligeable l'arbre de recherche de DP. Cette technique proposée initialement dans C-SAT [Dubois *et al.* 1996] a été reprise et adaptée dans de nombreuses implantations de DP (Satz [Li 1996, Anbulagan 1998], POSIT [Freeman 1995] etc.).

7.1.1.7 LOS (Locally Optimized Solution)

La notion de « *Locally Optimized Solution* » (LOS) a été introduite dans [Boufkhad 1996]. Elle correspond à une généralisation des notions de NPS et de PPS que nous reprenons ci-dessous :

Définition 7.1 (inversible, NPS et PPS)
Pour un modèle M, on dit que la variable v n'est pas **inversible** *si et seulement si l'interprétation I construite telle que : $\forall\, v' \neq v, I(v') = M(v')$ et $I(v) = \neg(M(v))$, n'est pas un modèle.*
Un **NPS (**« ***Negatively Prime Solution*** »**)** *est un modèle M tel que pour toute variable v si $M(v) = \mathbb{F}$ alors v n'est pas inversible.*
Un **PPS (**« ***Positively Prime Solution*** »**)** *est un modèle M tel que pour toute variable v si $M(v) = \mathbb{V}$ alors v n'est pas inversible.*

Étant donné un sous-ensemble $\mathcal{S}_L$ des littéraux de Σ, un LOS est un modèle de la formule $\Sigma \cup Contraintes$. $Contraintes$ est un ensemble de clauses permettant d'assurer que les littéraux de $\mathcal{S}_L$ ne sont pas inversibles. Plus précisément, $Contraintes$ se construit de la manière suivante :
$\forall x \in \mathcal{S}_L$:

- si x apparaît dans une seule clause $c = \{x \vee l_1 \vee l_2 \vee \ldots \vee l_n\}$ de Σ, alors ajouter à $Contraintes$ les clauses : $(\sim x \vee \sim l_1), (\sim x \vee \sim l_2), \ldots, (\sim x \vee \sim l_n)$;
- si x apparaît dans deux clauses $c_1 = \{x \vee l_1 \vee l_2 \vee \ldots \vee l_n\}$ et $c_2 = \{x \vee l'_1 \vee l'_2 \vee \ldots \vee l'_m\}$ de Σ, alors ajouter à $Contraintes$ les clauses : $(\sim x \vee \sim l_1 \vee \sim l'_1), (\sim x \vee \sim l_1 \vee \sim l'_2), \ldots, (\sim x \vee \sim l_1 \vee \sim l'_m), \ldots$, $(\sim x \vee \sim l_n \vee \sim l'_1), \ldots, (\sim x \vee \sim l_n \vee \sim l'_m)$;
- etc.

L'ajout de $Contraintes$ à Σ permet de réduire l'ensemble des modèles aux seuls PPS (ou NPS). Par ailleurs, Σ et $\Sigma \cup Contraintes$ sont équivalents du point de vue de la satisfaisabilité.

1. $\Sigma_l = (\Sigma \cup \{l\})$ (cf. définition 2.36).

Les longueurs des clauses ajoutées sont directement dépendantes du nombre d'occurrences des littéraux de $\mathcal{S}_L$. Il est alors préférable de choisir les littéraux ayant le moins d'occurrences.

Cette technique à base d'ajouts de clauses (limitée à l'ajout de clauses binaires et ternaires) a été intégrée à DP dans [Boufkhad 1996].

Cette simplification mise sur l'ajout de nouvelles contraintes pour simplifier l'arbre de résolution décrit par DP. Elle s'est avérée avoir un faible impact pour les instances *k***SAT** aléatoires au seuil, mais elle permet une réduction significative de l'arbre de recherche sur certains « *benchmarks* » de DIMACS [DIM1993].

7.1.1.8 Évaluation sémantique

L'évaluation sémantique [Jeannicot *et al.* 1988, Oxusoff & Rauzy 1989] est le nom donné à une variante de DP basée sur le théorème de partition du modèle (cf. théorème 7.1 page 109).

L'application de ce théorème permet une simplification, par élagage, de l'arbre de recherche de DP. L'idée est de couper les branches inexplorées de l'arbre de recherche entre deux nœuds (non nécessairement successifs) représentant deux ensembles de clauses Σ_1 et Σ_2 tels que $\Sigma_2 \subseteq \Sigma_1$.

Nous proposons dans la section 7.2 une généralisation du théorème de partition du modèle. Nous détaillons donc ce théorème dans le paragraphe 7.2.1.

7.1.2 Conditions de terminaisons

L'algorithme de Davis & Putnam possède deux conditions d'arrêt :

1. une testant si la base contient la clause vide, auquel cas l'ensemble de clauses est inconsistant ;
2. une seconde testant si l'ensemble de clauses est réduit à l'ensemble vide, auquel cas la base est consistante.

Ces deux tests sont appelés test d'inconsistance (respectivement de consistance) élémentaire. Cependant, il existe d'autres tests permettant la détection plus ou moins rapide de la satisfaisabilité ou de la non satisfaisabilité d'un ensemble de clauses. Nous présentons les techniques les plus répandues.

7.1.2.1 Tests d'inconsistance

Le test d'inconsistance élémentaire consiste à tester la présence de la claude vide. Or il est clair que cette clause vide n'a pu être produite que par l'affectation d'un littéral unitaire dont le complémentaire est également un littéral unitaire. Par conséquent le test suivant s'avère être plus intéressant :

```
if (Σ contient deux littéraux unitaires complémentaires) then return false ;
```

Remplacer le test d'inconsistance élémentaire dans DP et dans la procédure de propagation unitaire par cette nouvelle condition permet d'éviter la propagation de clauses unitaires inutiles en détectant plus tôt l'inconsistance.

Exemple 7.2 (Inconsistance par complémentarité)
Considérons la base construite sur 26 variables propositionnelles et constituée de 27 clauses unitaires : $\{\{a\} \wedge \{b\} \wedge \{c\} \wedge \ldots \wedge \{z\} \wedge \{\neg z\}\}$.

L'ordre de propagation unitaire étant le plus souvent arbitraire, si l'ordre alphabétique est choisi, il faut 26 propagations unitaires pour détecter l'inconsistance, alors que le test d'inconsistance par complémentarité détecte l'inconsistance sans aucune propagation.

Ce test d'inconsistance par complémentarité ne coûte pas plus cher que le test d'inconsistance élémentaire, ce qui explique son usage très répandu (à notre connaissance, toutes les meilleures implantations de DP utilisent ce test).

7.1.2.2 Tests de consistance

Le test élémentaire de consistance consiste à vérifier si toutes les clauses de l'ensemble initial de clauses sont satisfaites, c'est-à-dire à tester si la formule est réduite à l'ensemble vide de clause. Or une manière de prouver qu'un ensemble de clauses est satisfaisable est de trouver une interprétation partielle telle que tout prolongement de cette interprétation soit un modèle, c'est-à-dire de trouver un modèle partiel.

Par exemple, si la base Σ ne contient pas de clauses positives (respectivement négatives), un modèle est facilement obtenu en prolongeant l'interprétation partielle I_p (lue sur la branche) vers le modèle M tel que pour toute variable propositionnelle v : $M(v) = I_p(v)$ si $v \in I_p$ et $M(v) = \mathbb{F}$ (respectivement $\mathbb{V}$) sinon. Pour provoquer l'application du test, des heuristiques de branchement ont été introduites pour éliminer en priorité les clauses positives (respectivement négatives) [Hooker & Vinay 1995].

Dans [Lozinskii 1993], une autre technique permettant de détecter à l'avance de la consistance est développée. Cette méthode consiste à approcher le nombre d'interprétations falsifiant l'ensemble de clauses. Si ce nombre est inférieur au nombre d'interprétations possibles, alors la base est satisfaisable.

Propriété 7.6 (cf. [Lozinskii 1993])
Un ensemble de clauses Ω est satisfaisable si $\sum_{c \in \Omega} 2^{-l(c)} < 1$.

Ainsi le test de consistance élémentaire est remplacé par :

```
if ( Σ_{c∈Σ} 2^{-l(c)} < 1 ) then return true ;
```

Toutefois, les techniques exposées sont souvent trop coûteuses en temps et par ailleurs, leur application ne permet pas de couper un nombre significatif de branches. Par conséquent, ces techniques entraînent la plupart du temps un ralentissement de la procédure DP et ne sont donc pas utilisées en pratique.

7.1.3 Heuristiques de branchement

La fonction `Heuristique_de_branchement` est déterminante pour l'efficacité de la procédure DP. Les incidences de l'heuristique sur la taille de l'arbre et par conséquent sur les temps d'exécution sont considérables. Il ne faut donc pas négliger cette fonction.

Deux approches complémentaires sont utilisées pour l'élection du prochain littéral : sémantiques et syntaxiques.

7.1.3.1 Approches « sémantiques »

La littérature comporte peu d'approches sémantiques pour le calcul propositionnel (*e.g.* [Crawford 1993, Mazure *et al.* 1996a, Mazure *et al.* 1998a]). Ces méthodes utilisent des solveurs incomplets pour extraire une information « sémantique » permettant d'établir une stratégie de branchement. Nous détaillons les stratégies que nous avons proposées [Mazure *et al.* 1996a, Mazure *et al.* 1998a] dans le chapitre 8. Il est à noter que ces techniques permettent de diminuer de manière significative l'arbre de recherche construit par DP sur un grand nombre d'instances et en conséquence de réduire le temps d'exécution de manière non négligeable.

7.1.3.2 Approches « syntaxiques »

L'approche syntaxique consiste à effectuer le choix du littéral à l'aide d'opérations plus ou moins complexes portant exclusivement sur la syntaxe. Plus précisément, les éléments syntaxiques pris en compte se résument aux nombres d'occurrences des littéraux et à la taille des clauses où ils apparaissent. On comprend aisément l'importance du nombre d'occurrences d'un littéral : plus un littéral apparaît dans la base, plus il va satisfaire de clauses et plus il raccourcira de clauses lors de la branche correspondante à l'affectation de son complémentaire. Par ailleurs, raccourcir les clauses les plus courtes permet de produire plus de clauses unitaires nécessaires à la simplification du problème. La taille des clauses est donc un facteur important dans le calcul du meilleur littéral.

Contrairement aux approches sémantiques, les heuristiques syntaxiques se calculent souvent très rapidement : l'analyse de la forme se faisant la plupart du temps plus rapidement que l'analyse du sens. Cependant, cet avantage peut parfois se révéler défavorable dans certains cas. En effet, il est simple de « cacher » le meilleur littéral par l'intermédiaire d'artifices syntaxiques (*e.g.* ajout de clauses sous-sommées, ajout de littéraux impliqués, etc.).

Les heuristiques syntaxiques les plus performantes possèdent comme point commun de ne pas dissocier un littéral de son complémentaire. Ceci s'explique par le fait qu'il est préférable d'avoir des arbres équilibrés. En effet, il est préférable d'explorer deux sous-arbres de taille 2^{k-1} que deux sous-arbres de taille 2^{k-2} et 2^k. Les meilleures fonctions d'évaluations tendent à choisir comme prochaine variable celle dont l'affectation produit deux sous-arbres de taille (estimée) similaire (cf. description page suivante).

De manière générale, la fonction `Heuristique_de_branchement` s'écrit de la façon suivante :

Algorithme 7.4 HEURISTIQUE DE BRANCHEMENT

```
1.  Function HEURISTIQUE_DE_BRANCHEMENT : un ensemble de clauses
2.  Input  : Σ un ensemble de clauses sans clause unitaire et sans littéral pur ;
3.  Output : l le prochain littéral à affecter
4.  Begin
5.    foreach variable v de Σ
6.    do
7.       Calculer Poids_lit(v) ;
8.       Calculer Poids_lit(¬v) ;
9.       score[v] = F(Poids_lit(v),Poids_lit(¬v)) ;
               %% le score de chaque variable est évalué en fonction des poids de v et ¬v
10.   done
11.   v = la variable ayant le meilleur score ;
12.   return v ou ¬v suivant un critère de sélection de signe ;
13. End
```

La fonction d'évaluation : Les fonctions (F) permettant de calculer le score d'une variable foisonnent dans la littérature ([Dubois *et al.* 1996, Freeman 1995, Crawford & Auton 1996, Silva & Sakallah 1996b, Li 1996, Li & Anbulagan 1997, Bayardo Jr. & Schrag 1997], etc.). Ces fonctions sont toutes singulières. Cependant afin d'équilibrer au maximum l'arbre de recherche, toutes ces fonctions tiennent compte à la fois du poids (cf. la fonction poids ci-après) du littéral et du poids de son complémentaire.

Le critère de signe : Dans le cas, où l'instance est inconsistante, le signe du littéral (positif ou négatif) retourné est sans importance puisque les deux branches seront explorées. À l'inverse, si l'on cherche à spécialiser l'heuristique dans la recherche de solutions ce choix est primordial puisqu'il peut permettre d'éviter l'exploration d'une des branches dans le cas d'un instance satisfaisable. Les critères souvent retenus sont le nombre d'occurrences des littéraux :

– if (Occ(l) > Occ($\neg l$)) then return l else return $\neg l$ fi

ou des critères faisant intervenir les poids des littéraux :

– if ($Poids_{lit}(l) > Poids_{lit}(\neg l)$) then return l else return $\neg l$ fi

L'objectif de ces critères est l'exploration prioritaire de la branche qui possède la plus forte probabilité d'aboutir à un modèle. Il existe de nombreux critères plus raffinés (*e.g.* [Dubois *et al.* 1996], [Freeman 1995], [Anbulagan 1998], etc.) mais utilisant pour la plupart les poids des littéraux.

La fonction poids : La fonction de calcul du poids d'un littéral ($Poids_{lit}$) s'écrit généralement sous la forme : $Poids_{lit}(l) = \sum_{l \in c} Poids_{cla}(c)$
Le calcul de $Poids_{cla}$ fait généralement intervenir la longueur de la clause.
Comme pour la fonction F, de nombreuses fonctions de calcul du poids d'une clause ont été proposées. Nous

présentons les plus employées. En outre, on trouve dans la littérature des combinaisons de ces fonctions poids.

- $Poids_{cla}(c) = 1$ si $l(c) = 2$, 0 sinon. Seules les clauses binaires sont prises en compte dans l'heuristique [Billionnet & Sutter 1992] ;
- une généralisation du poids précédent est : $Poids_{cla}(c) = 1$ s'il n'existe pas de clause de taille inférieure à $l(c)$, 0 sinon ;
- $Poids_{cla}(c) = \frac{1}{2^{l(c)}}$. Cette fonction a été proposée par Jeroslow & Wang [Jeroslow & Wang 1990]. Elle permet de lier le poids de la clause avec la probabilité qu'une interprétation donnée falsifie la clause c ($Poids_{cla}(c) = \frac{1}{2^{l(c)}} = \frac{2^{V-l(c)}}{2^V}$ où V représente le nombre de variables du problème). Cette fonction est très fréquemment utilisée et fait partie des fonctions donnant les meilleurs résultats ;
- $Poids_{cla}(c) = \frac{1}{\gamma^{l(c)}}$ où γ est une constante. Cette fonction est clairement une généralisation de l'heuristique de Jeroslow-Wang (JW). Par ailleurs, plus γ augmente, plus la fonction $Poids_{cla}$ accentue la différence entre les clauses de différentes longueurs. Par exemple, Freeman [Freeman 1995] utilise dans POSIT une valeur de γ fixée à 5.
- $Poids_{cla}(c) = \frac{1}{2^{l(c)}}$ si c est une clause raccourcie, 0 sinon. Ce poids est un raffinement de celui proposé par Jeroslow & Wang, il permet de se concentrer sur les clauses déja « touchées ». Cette heuristique donne de bons résultats en moyenne et favorise l'application du théorème de partition du modèle. Elle fut proposée intialement dans [Rauzy 1994] sous le nom de « *First Fail In Shortened* (ffis)». Dans cette heuristique afin de départager les littéraux ayant le même score, l'heuristique JW traditionnelle est utilisée.
- $Poids_{cla}(c) = -\ln(1 - \frac{1}{(2^{l(c)}-1)^2})$. Cette fonction a été proposée dans [Dubois *et al.* 1996] pour l'algorithme C-SAT. Elle est reconnue, à l'heure actuelle, comme étant celle qui donne les meilleurs résultats sur les instances *3***SAT** aléatoires au seuil.

Remarque 7.2
Nous avons vu que la plupart des heuristiques utilisent $Poids_{lit}(l) = \sum_{l \in c} Poids_{cla}(c)$, comme fonction d'évaluation du poids d'un littéral. L. Brisoux *et al.* proposent d'ajouter un facteur multiplicatif $\gamma(c)$ au poids des clauses : $Poids_{lit}(l) = \sum_{l \in c} \gamma(c) Poids_{cla}(c)$ [Brisoux *et al.* 1998]. Pour chaque clause c de la base, $\gamma(c)$ est initialisé à 1 au début de la recherche, dès qu'une clause devient vide (inconsistante), $\gamma(c)$ est augmenté de 1. La greffe de cette technique aux heuristiques traditionnelles permet d'obtenir dans un grand nombre de cas une réduction de la taille de l'arbre de recherche décrit par DP.

7.1.4 Meilleures implantations de DP

Cette section présente les implantations de DP reconnues comme les plus efficaces en pratique actuellement. Nous ne souhaitons pas, au travers de ce paragraphe porter un quelconque jugement de valeur sur ces méthodes. Par ailleurs, la liste dressée se veut non exhaustive. En effet, d'autres implantations de DP sont au moins aussi efficaces en pratique que celles présentées dans cet état de l'art.(*e.g.* SATO [Zhang & Stickel 1996], etc.).

Nous présentons ci-après, les principales fonctions de simplification utilisées par ces différentes implantations ainsi que les caractéristiques de leur heuristique de branchement. Cependant le lecteur intéressé pourra trouver de plus amples détails dans les références citées pour chacune des implantations.

7.1.4.1 C-SAT

C-SAT a été proposé initialement dans le but d'améliorer de façon sensible l'efficacité des méthodes de résolution de **SAT** en les spécialisant soit dans la recherche d'une solution, soit dans la preuve de l'inexistance de solution. Cette idée est justifiée d'une part par l'existence des phénomènes de seuil qui séparent de façon tranchée les instances satisfaisables des instances insatisfaisables et d'autre part par des manières très différentes d'explorer efficacement l'espace potentiel des solutions selon que l'on suppose qu'il existe ou non une solution.

C-SAT est un système de résolution du problème SAT basé sur DP, programmé en « C ANSI » et spécialisé dans la preuve de l'inexistence de solution. Cependant, C-SAT s'avère également être extrêmement efficace sur les instances satisfaisables. Il est à noter que C-SAT est le premier algorithme complet à avoir été capable de résoudre des instances aléatoires au seuil de 500 variables en un temps raisonnable (< 24 heures en moyenne). C-SAT est principalement constitué :

- d'un pré-traitement à la racine de l'arbre de DP intégrant la détection de symétries élémentaires, la production de clauses binaires via une forme de résolution et la production des résolvantes de longueur inférieure ou égale aux clauses qui se résolvent ;
- d'une heuristique de branchement permettant une approximation de la profondeur de l'arbre de DP ;
- d'une méthode de simplification locale appelée « *local processing* » permettant de fixer des littéraux en utilisant la propagation unitaire.

L'heuristique de branchement de C-SAT maximise la fonction score suivante pour $i = 2$:
Pour toute variable x de la formule CNF Ω :

$$Score_i(x) = 1.5 \min(F_i(x), F_i(\neg x)) + F_i(x) + F_i(\neg x)$$

où

$$F_i(x) = f(x) + \sum_{\{x \vee y\} \in \Omega} F_{i-1}(\neg y)$$

et

$$F_0(x) = f(x) - W_2$$

La fonction $f(x)$ s'écrit sous la forme : $f(x) = \sum_{x \in c} W_{l(c)}$ où $W_{l(c)}$ correspond à la fonction poids de la clause c :

$$W_{l(c)} = -\ln(1 - \frac{1}{(2^{l(c)} - 1)^2})$$

Une description complète et détaillée de C-SAT peut être trouvée dans [Dubois *et al.* 1996, Boufkhad 1996].

7.1.4.2 POSIT

POSIT (*PrOpositional SatIsfiability Testbed)* est une implantation de DP proposée par Jon William Freeman [Freeman 1996]. Ce logiciel est écrit en « C ANSI ». La description complète de POSIT peut être trouvée dans [Freeman 1995].

La simplification réalisée à la racine de l'arbre de DP s'effectue en quatre étapes :

1. la propagation unitaire ;

2. la simplification par les littéraux purs ;
3. l'élimination des variables x tel que x ou $\neg x$ soit un singleton ;
4. l'élimination des variables x tel que x ou $\neg x$ soit un doublet ;

Si les étapes 3. et 4. créent un littéral unitaire ou pur, les étapes 1. et 2. sont répétées. Il est à noter qu'après ces simplifications chaque proposition apparaît au moins 6 fois : 3 fois positivement et 3 fois négativement.

Un singleton (respectivement doublet) est une variable telle que son nombre d'occurrences positives ou négatives est égal à 1 (respectivement à 2). L'élimination des singletons et des doublets l s'effectue par la production de l'ensemble des résolvantes en l. Plus généralement si une variable l apparaît positivement dans les clauses $(l \vee C_1), (l \vee C_2), \ldots, (l \vee C_k)$ et négativement dans les clauses $(\neg l \vee C'_1), (\neg l \vee C'_2), \ldots, (\neg l \vee C'_{k'})$ où C_i et C'_j sont des clauses, alors ces clauses sont remplacées par l'ensemble de clauses :

$$\bigwedge_{i=1}^{k} \bigwedge_{j=1}^{k'} (C_i \vee C'_j)$$

Cette technique de simplification a été proposée initialement dans [Franco 1991, Dechter & Rish 1994].

L'heuristique de branchement procède en 4 étapes principales :

1. la sélection d'un sous-ensemble $\mathcal{S}$ de variables en fonction de leur nombre d'occurrences dans les clauses binaires ;
2. la propagation une à une des variables de $\mathcal{S}$ (positivement puis négativement), afin d'estimer leur score en terme du nombre de nouvelles clauses binaires produites ;
3. la suppression des variables de $\mathcal{S}$ n'ayant pas obtenu pas le meilleur score ;
4. l'élection d'un littéral parmi ceux de $\mathcal{S}$ en utilisant comme poids pour les littéraux : $Poids_{lit} = \frac{1}{5^{l(c)}}$ (si $\mathcal{S} = \varnothing$, le littéral est élu par rapport à l'ensemble complet des littéraux de la formule).

7.1.4.3 Tableau

Tableau est une version de DP proposée par James M. Crawford et Larry D. Auton. Tableau est écrit en « C ANSI » et existe sous la forme de deux programmes *ntab* et *3tab*, *3tab* étant spécialisé dans le traitement d'instances *3***SAT**. Initalement Tableau est une implantation de la méthode des tableaux de R. M. Smullyan [Smullyan 1968].

De manière générale, Tableau intègre une structure de données judicieuse permettant d'effectuer la propagation des clauses unitaires de façon très efficace. Le poids des littéraux, pour l'heuristique de branchement approxime le nombre de clauses unitaires engendrées par l'affectation du littéral en question. Cette heuristique est similaire à celle proposée dans [Zabih & McAllester 1988] et celle utilisée dans POSIT [Freeman 1995]. Le score d'une variable x est calculé par la fonction suivante :

$$score(x) = Poids_{lit}(x) \times Poids_{lit}(\neg x) \times 1024 + Poids_{lit}(x) + Poids_{lit}(\neg x) + 1$$

où $Poids_{lit}(l)$ est le nombre d'occurrences du littéral l dans les clauses binaires.

Ce score permet de sélectionner un sous-ensemble de variables pour lesquelles la propagation unitaire est effectuée. Le littéral retourné par l'heuristique est celui qui aura engendré par sa propagation le plus grand nombre de clauses binaires. Si l'on note N le nombre de variables initialement dans la CNF et n le nombre de variables restant à un nœud donné, le nombre de variables examinées est égal à $N - 21 \times n$.

Une description détaillée de Tableau peut être trouvée dans [Crawford & Auton 1996].

7.1.4.4 Satx et Satz

Satx est un système de résolution basé sur DP proposé par ChuMin Li. Il intègre à chaque point de choix la propagation de toutes les « variables fortement contraintes » [Li 1996].

Définition 7.2 (Variables fortement contraintes)
Une variable est **fortement contrainte** *pour une CNF si et seulement si elle apparaît positivement et négativement dans les clauses binaires et que son nombre d'occurrences dans les clauses binaires est supérieur ou égal à trois.*

Lors d'un point de choix, l'heuritique de Satx choisit le littéral qui parmi les variables les plus fortement contraintes, maximise le nombre de clauses binaires produites lors de sa propagation.

Satz est une extension de Satx proposée par [Anbulagan 1998]. Il diffère de Satx principalement par trois caractéristiques :

1. la production des résolvantes de taille inférieure ou égale à trois en pré-traitement ;
2. l'intégration du nombre de résolvantes binaires dans l'heuristique de branchement ;
3. l'extension du nombre de variables examinées à chaque point de choix suivant une propriété appelée $PROP_z$.

Satz est capable de résoudre des instances *3***SAT** aléatoires de 500 variables au seuil. Les descriptions détaillées de ces systèmes peuvent être trouvées dans [Anbulagan 1998, Li & Anbulagan 1997, Li 1996].

7.1.4.5 GRASP

GRASP est un système de résolution de **SAT** initialement spécialisé pour le traitement d'instances issues du monde réel. Il est programmé en « C++ ». GRASP est une variante de DP, et se distingue des autres implantations de DP par des procédures puissantes d'analyse de conflits.

L'analyse de conflits est utilisée pour élaguer au maximum l'arbre de recherche. On trouve dans GRASP, trois méthodes différentes de diagnostic de conflits :

1. « *Conflict-Directed Backtracking* » qui par l'analyse de la provenance de l'inconsistance permet un « backtrack » non chronologique ;
2. « *Conflict-Based Equivalence* » identifie les conditions suffisantes pour que deux branches de l'arbre aboutissent au même conflit. Cette méthode permet d'opérer le « backtrack » plus tôt sans identifier explicitement l'inconsistance ;
3. « *Failure-Driven Assertions* » dénote les valeurs d'affectation des variables nécessaires à l'identification d'un conflit.

Une description détaillée de GRASP peut être trouvée dans [Silva & Sakallah 1996b]. Par ailleurs, dans [Silva & Sakallah 1996a], les auteurs montrent, sur de nombreuses instances structurées et issues du monde réel, l'efficacité pratique des techniques d'analyse de conflits.

7.1.4.6 relsat(4)

Roberto J. Bayardo Jr. and Robert C. Schrag proposent dans [Bayardo Jr. & Schrag 1997] plusieurs versions de DP intégrant différentes techniques sophistiquées développées initialement pour les CSP (*e.g.* « *Conflict-Directed backtracking*[2] », « *relevance-bounded learning* », etc.).

Comparativement aux autres implantations, relsat(4) s'appuie sur le principe de « *relevance-bounded learning* » défini dans [Bayardo Jr & Miranker 1996] qui consiste à ajouter des résolvantes lors du retour arrière[3]. Cet algorithme émerge par ses performances en pratique sur de nombreuses instances structurées issues d'applications diverses (diagnostic, planification, etc.).

La description détaillée de relsat(4) est réalisée dans [Bayardo Jr. & Schrag 1997].

7.2 Partition du modèle

7.2.1 Définition

Le théorème de partition du modèle, utilisé dans la méthode de l'évaluation sémantique permet d'éliminer des branches de l'arbre de recherche de DP [Jeannicot *et al.* 1988, Oxusoff & Rauzy 1989]. Il s'exprime de la façon suivante :

Note 7.1
- Nous rappelons que $\Sigma_{\{l_1,\ldots,l_j\}}$ est l'ensemble de clauses Σ simplifié successivement par $l_1, l_2, \ldots, l_j$.
- Pour plus de commodité de lecture nous notons $\Sigma_{l_{[i:j]}}$ l'ensemble de clauses $\Sigma_{\{l_i,\ldots,l_j\}}$.

Théorème 7.1 (Partition du modèle [Oxusoff & Rauzy 1989])
Soient Σ un ensemble de clauses et $\{l_1, \ldots, l_j, \ldots, l_k\}$ un ensemble fondamental de littéraux.
Si $\Sigma_{l_{[1:k]}} \subseteq \Sigma_{l_{[1:j]}}$, alors $\Sigma_{l_{[1:j]}}$ est satisfaisable si et seulement si $\Sigma_{l_{[1:k]}}$ est satisfaisable.

Exemple 7.3 (Exemple d'inclusion de base de clauses)
Soient $\Sigma = \{(l_1 \vee l_2 \vee \neg l_4), (\neg l_1 \vee l_4), (\neg l_1 \vee l_2 \vee l_3)\}$ un ensemble de clauses et $\{l_1, \neg l_2, l_3\}$ un ensemble de littéraux, nous avons :

$$\left.\begin{array}{l}\Sigma_{l_{[1:1]}} = \{(l_4), (l_2 \vee l_3)\}\\ \Sigma_{l_{[1:2]}} = \{(l_4), (l_3)\}\\ \Sigma_{l_{[1:3]}} = \{(l_4)\}\end{array}\right\} \Longrightarrow \left\{\begin{array}{lll}\Sigma_{l_{[1:3]}} & \subset & \Sigma_{l_{[1:2]}}\\ \Sigma_{l_{[1:3]}} & \subset & \Sigma_{l_{[1:1]}}\\ \Sigma_{l_{[1:3]}} & \not\subset & \Sigma\\ \Sigma_{l_{[1:2]}} & \not\subset & \Sigma_{l_{[1:1]}}\\ \Sigma_{l_{[1:2]}} & \not\subset & \Sigma\\ \Sigma_{l_{[1:1]}} & \not\subset & \Sigma\end{array}\right.$$

S. Jeannicot, L. Oxusoff et A. Rauzy proposent une application directe du théorème de partition du modèle au sein de la procédure de Davis & Putnam. En effet, dès qu'une inclusion est détectée entre deux nœuds de l'arbre, les branches situées entre ces deux nœuds sont coupées. Ils appellent *évaluation sémantique* la procédure de Davis & Putnam associée au théorème de partition du modèle. Nous illustrons cette

2. Appelé également « *conflict directed backjumping* ».
3. Il est à noter qu'indépendemment Thierry Castell a proposé une méthode similaire dans [Castell 1997]

méthode sur l'exemple proposé dans [Oxusoff & Rauzy 1989] (cf. exemple 7.4)

Exemple 7.4 (Arbre de recherche de l'évaluation sémantique)
Soit $\Sigma = \{(l_1 \vee l_2 \vee \neg l_4), (\neg l_1 \vee l_4), (\neg l_1 \vee l_2 \vee l_3), (\neg l_2), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\}$.

L'arbre représenté dans la figure 7.3 page ci-contre est obtenu par la procédure de Davis & Putnam utilisant l'heuristique de branchement retournant un littéral positif dans l'ordre lexicographique et le théorème de partition du modèle. Par ailleurs, pour des raisons de lisibilité, les simplifications par littéraux unitaires et purs ne sont pas mentionnées.

Dans cet arbre, nous avons :

$$\begin{array}{lcl}
\Sigma_{\{l_1\}} & = & \{(l_4), (l_2 \vee l_3), (\neg l_2), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\} \\
\Sigma_{\{l_1, \neg l_2\}} & = & \{(l_4), (l_3), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\} \\
\Sigma_{\{l_1, \neg l_2, l_3\}} & = & \{(l_4), (l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4\}} & = & \{(l_5 \vee l_6), (l_5 \vee \neg l_6), (\neg l_5 \vee l_6), (\neg l_5 \vee \neg l_6)\} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4, l_5\}} & = & \{(l_6), (\neg l_6)\} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4, \neg l_5\}} & = & \{(l_6), (\neg l_6)\}
\end{array}$$

et par conséquent, nous obtenons les inclusions suivantes :

$$\left.\begin{array}{lcl}
\Sigma_{\{l_1, \neg l_2, l_3\}} & \subset & \Sigma_{\{l_1, \neg l_2\}} \\
\Sigma_{\{l_1, \neg l_2, l_3\}} & \subset & \Sigma_{\{l_1\}}
\end{array}\right) \text{ Première application du théorème de partition du modèle}$$

$$\left.\begin{array}{lcl}
\Sigma_{\{l_1, \neg l_2, l_3, l_4\}} & \subset & \Sigma_{\{l_1, \neg l_2, l_3\}} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4\}} & \subset & \Sigma_{\{l_1, \neg l_2\}} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4\}} & \subset & \Sigma_{\{l_1\}} \\
\Sigma_{\{l_1, \neg l_2, l_3, l_4\}} & \subset & \Sigma
\end{array}\right) \text{ Seconde application du théorème de partition du modèle}$$

Remarque 7.3
Afin de clore ce rappel sur le théorème de partition du modèle, nous terminons par trois remarques.

1° La première remarque porte sur le lien naturel existant entre « partition du modèle » et « simplification par littéraux purs ». De manière évidente, la partition du modèle généralise la règle de simplification par les littéraux purs. En effet, si l est un littéral pur d'une CNF Σ, l'opération de simplification de Σ par l supprime de Σ toutes les clauses contenant l et laisse intactes toutes les autres clauses. Par conséquent, il est clair que $\Sigma_l \subseteq \Sigma$. Ainsi l'application du théorème de partition du modèle permet de se passer pour DP de l'étape de simplification par les littéraux purs.

2° La seconde remarque porte sur la recherche de la plus grande partition d'un ensemble de clauses. Détecter le plus petit i tel que $\Sigma_{l_{[1:k]}}$ $(k \geqslant i)$ soit inclus dans $\Sigma_{l_{[1:i]}}$, peut paraître un processus complexe et coûteux en temps. Or la propriété suivante permet d'obtenir une méthode de détection incrémentale, tout en garantissant l'obtention du plus grand $\Sigma_{l_{[1:i]}}$.

Propriété 7.7 ([Oxusoff & Rauzy 1989])
Si $\Sigma_{l_{[1:k]}} \subseteq \Sigma_{l_{[1:i]}}$, *alors* $\Sigma_{l_{[1:k]}}$ *est également inclus dans* $\Sigma_{l_{[1:i+1]}}, \Sigma_{l_{[1:i+2]}}, \ldots, \Sigma_{l_{[1:k-1]}}$.
Ce qui s'exprime encore sous la forme :
Si $\Sigma_{l_{[1:k]}} \not\subset \Sigma_{l_{[1:i]}}$, *alors* $\Sigma_{l_{[1:k]}}$ *n'est pas inclus dans* $\Sigma_{l_{[1:i-1]}}, \Sigma_{l_{[1:i-2]}}, \ldots, \Sigma_{l_{[1:1]}}, \Sigma$.

Ces propriétés permettent d'éviter tous les tests d'inclusion inutiles. L'efficacité de la détection du plus grand ensemble de clauses incluant $\Sigma_{l_{[1:k]}}$ dépend donc fortement de l'efficacité du test d'inclusion entre $\Sigma_{l_{[1:k]}}$ et $\Sigma_{l_{[1:i]}}$ $(k \geqslant i)$. Or ce test est relativement rudimentaire, il consiste à vérifier que toutes les clauses de $\Sigma_{l_{[1:i]}}$ contenant un littéral parmi l'ensemble $\{\sim l_{i+1}, \sim l_{i+2}, \ldots, \sim l_k\}$

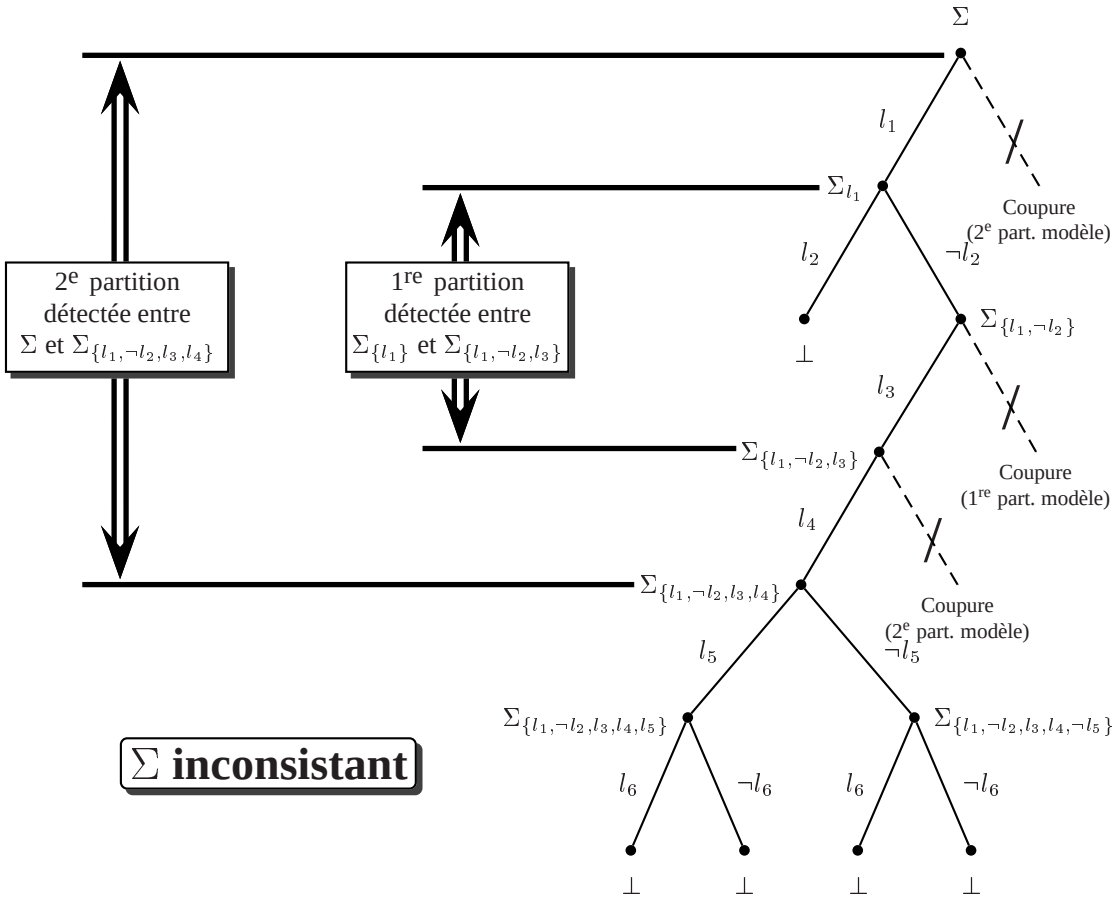

FIG. 7.3 – *Arbre de recherche construit par l'algorithme d'évaluation sémantique*

n'apparaissent pas (sont satisfaites) dans $\Sigma_{l_{[1:k]}}$. En utilisant un codage judicieux des ensembles de clauses, ce test peut être effectué de manière efficace. Ainsi, détecter la plus grande partition est une opération qui peut être réalisée à moindre coût.

3° La dernière remarque porte sur la détection de l'ensemble de toutes les inclusions possibles lors de l'exploration de DP. Afin de garantir la détection de toutes les inclusions, une étape de suppression des clauses sous-sommées est nécessaire. Nous illustrons notre propos sur l'exemple suivant.

Exemple 7.5 (Partition du modèle et sous-sommation)
Soit $\Sigma = \{(\neg l_1 \vee l_3), (l_2 \vee l_4), (\neg l_2 \vee l_3 \vee l_4)\}$, nous avons :

$\Sigma_{l_{[1:1]}} = \{(l_3), (l_2 \vee l_4), (\neg l_2 \vee l_3 \vee l_4)\}$
$\Sigma_{l_{[1:2]}} = \{(l_3), (l_3 \vee l_4)\}$

Dans ce cas, aucune inclusion n'est détectée. Par contre, si $\Sigma_{l_{[1:2]}}$ est simplifiée par la suppression des clauses sous-sommées, alors $\Sigma_{l_{[1:2]}}$ devient égale à $\Sigma'_{l_{[1:2]}} = \{(l_3)\}$ et nous obtenons : $\Sigma'_{l_{[1:2]}} \subset \Sigma_{l_{[1:1]}}$ qui provoque l'application du théorème de partition du modèle.

Il reste que la simplification des clauses sous-sommées est une opération extrêmement coûteuse en temps (cf. paragraphe 7.1.1.3 page 98). Par conséquent, détecter toutes les inclusions nuit fortement à l'efficacité de la procédure d'évaluation sémantique. Cependant, il est à noter que l'algorithme d'évaluation sémantique demeure valide même si la simplification des clauses sous-sommées n'est pas réalisée. Mais il est alors impossible de garantir la détection de tous les cas

d'applications possibles du théorème de partition du modèle.

7.2.2 Généralisation

Nous proposons dans cette section une généralisation originale du théorème de partition du modèle. Cette généralisation porte sur le lien (l'inclusion pour le théorème de partition du modèle « classique ») entre les ensembles de clauses. En effet, nous montrons de quelle manière il est possible de remplacer l'inclusion ensembliste ($\subset$) par la conséquence sémantique ($\models$) de deux ensembles de clauses.

Théorème 7.2 (Partition du modèle généralisé)
Soient Σ un ensemble de clauses et $\{l_1, \ldots, l_j, \ldots, l_k\}$ un ensemble fondamental de littéraux.
Si $\Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}} (j \leqslant k)$, alors $\Sigma_{l_{[1:j]}}$ est satisfaisable si et seulement si $\Sigma_{l_{[1:k]}}$ est satisfaisable.

Preuve (Théorème 7.2)
$\boxed{\Rightarrow}$ Si $\Sigma_{l_{[1:j]}}$ est satisfaisable alors $\Sigma_{l_{[1:k]}}$ est satisfaisable, par définition de la conséquence sémantique (cf. définition 2.17 page 12).

$\boxed{\Leftarrow}$ Supposons $\Sigma_{l_{[1:k]}}$ satisfaisable et soit I un modèle de $\Sigma_{l_{[1:k]}}$. Montrons que I peut être prolongé vers un modèle I' de $\Sigma_{l_{[1:j]}}$.
Soit I' l'interprétation de $\Sigma_{l_{[1:j]}}$ telle que :

$$\text{pour tout littéral } l \text{ de } \Sigma_{l_{[1:j]}} : \begin{cases} \text{si } l \in \{l_{j+1}, l_{j+2}, \ldots, l_k\} & \text{alors} \quad I'(l) = \mathbb{V} \\ \text{sinon si } \sim l \in \{l_{j+1}, l_{j+2}, \ldots, l_k\} & \text{alors} \quad I'(l) = \mathbb{F} \\ \text{sinon} & \quad\quad\; I'(l) = I(l) \end{cases}$$

Montrons que I' est un modèle de $\Sigma_{l_{[1:j]}}$. Soit c une clause de $\Sigma_{l_{[1:j]}}$, trois cas sont envisageables :

- c ne contient pas de littéraux de $\{l_{j+1}, l_{j+2}, \ldots, l_k\}$, dans ce cas $c \in \Sigma_{l_{[1:k]}}$ donc I (modèle de $\Sigma_{l_{[1:k]}}$) satisfait c et ainsi par construction de I' : I' satisfait c ;
- c contient à la fois des littéraux appartenant et n'appartenant pas à $\{l_{j+1}, l_{j+2}, \ldots, l_k\}$, dans ce cas il existe une clause c' de $\Sigma_{l_{[1:k]}}$ telle que $c' \subset c$, ce qui signifie encore que c' sous-somme c ($c' \models c$). De plus, comme I satisfait c', I' satisfait c' par construction de I'. Ainsi $c' \models c$ et c' satisfait par I' impliquent que I' satisfait c ;
- c ne contient que des littéraux de $\{l_{j+1}, l_{j+2}, \ldots, l_k\}$, dans ce cas c est satisfait par I', sinon il existerait une clause vide c' dans $\Sigma_{l_{[1:k]}}$ telle que c' est extraite de c par l'interprétation des littéraux $l_{j+1}, l_{j+2}, \ldots, l_k$. Si une telle clause vide existe dans $\Sigma_{l_{[1:k]}}$, alors $\Sigma_{l_{[1:k]}}$ est inconsistant, ce qui est contradictoire avec notre hypothèse de départ. Par conséquent I' satisfait c.

Dans tous les cas, c est satisfait par I', par conséquent I' est un modèle de $\Sigma_{l_{[1:j]}}$ (CQFD). □

Il est clair que ce nouveau théorème généralise le théorème de partition du modèle. En effet, nous montrons que :

Propriété 7.8
Soient Σ et Ω deux ensembles de clauses satisfaisables. Si $\Sigma \subseteq \Omega$, alors $\Omega \models \Sigma$.

Preuve (Propriété 7.8)
Si $\Sigma \subset \Omega$, alors toutes les clauses de Σ appartiennent également à Ω. Tout modèle de Ω satisfait chacune des clauses de Ω, par définition d'un modèle (cf. définition 2.13). Par conséquent, tout modèle de Ω satisfait

chacune des clauses de Σ. Ainsi, tout modèle de Ω est un modèle de Σ, ce qui entraîne par définition de la conséquence logique que $\Omega \models \Sigma$. □

Remarque 7.4
Il faut remarquer que la réciproque est fausse ($(\Omega \models \Sigma) \not\Rightarrow (\Sigma \subseteq \Omega)$), comme l'illustre l'exemple ci-dessous.

Exemple 7.6 (Exemple de non réciprocité de la propriété 7.8)
Soient $\Omega = \{(a \vee b \vee c), (\neg a \vee b)\}$ et $\Sigma = \Omega_{\neg a} = \{(b \vee c)\}$, nous avons bien $\Omega \models \Sigma$ mais $\Sigma \not\subset \Omega$.

Par ailleurs, l'un des atouts majeurs du théorème de partition du modèle réside dans son utilisation incrémentale (cf. propriété 7.7). Cette propriété est conservée pour le théorème de partition du modèle généralisé.

Propriété 7.9
Si $\Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}$, *alors* $\Sigma_{l_{[1:k]}}$ *est également une conséquence logique* $\Sigma_{l_{[1:j+1]}}, \Sigma_{l_{[1:j+2]}}, \ldots, \Sigma_{l_{[1:k-1]}}$.

Preuve (Propriété 7.9)
Si $\Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}$, alors $(\Sigma_{l_{[1:j]}} \wedge \neg\Sigma_{l_{[1:k]}})$ est inconsistant. Donc a fortiori, $(\Sigma_{l_{[1:j]}} \wedge \neg\Sigma_{l_{[1:k]}} \wedge (l_{j+1}))$ est inconsistant. Or $(\Sigma_{l_{[1:j]}} \wedge (l_{j+1})) = (\Sigma_{l_{[1:j]}})_{l_{j+1}} = \Sigma_{l_{[1:j+1]}}$. Par conséquent $(\Sigma_{l_{[1:j+1]}} \wedge \neg\Sigma_{l_{[1:k]}})$ est inconsistant et donc $\Sigma_{l_{[1:j+1]}} \models \Sigma_{l_{[1:k]}}$. Un raisonnement identique est appliqué pour montrer que $\Sigma_{l_{[1:k]}}$ est une conséquence logique de $\Sigma_{l_{[1:j+2]}}, \ldots, \Sigma_{l_{[1:k-1]}}$. □

Un corollaire immédiat à cette propriété trouve une application directe en pratique comme nous le montrons dans la section suivante (cf. paragraphe 7.2.3).

Corollaire 7.1
Si $\Sigma_{l_{[1:j]}} \not\models \Sigma_{l_{[1:k]}}$, *alors* $\Sigma_{l_{[1:k]}}$ *n'est pas non plus une conséquence logique* $\Sigma_{l_{[1:j-1]}}, \Sigma_{l_{[1:j-2]}}, \ldots, \Sigma_{l_{[1:1]}}, \Sigma$.

Preuve (Corollaire 7.1)
Dans un premier temps, montrons que si $\Sigma_{l_{[1:j]}} \not\models \Sigma_{l_{[1:k]}}$, alors $\Sigma \not\models \Sigma_{l_{[1:k]}}$.

Supposons que $\Sigma_{l_{[1:j]}} \not\models \Sigma_{l_{[1:k]}}$ et que $\Sigma \models \Sigma_{l_{[1:k]}}$. Si $\Sigma \models \Sigma_{l_{[1:k]}}$, alors d'après la propriété 7.9, nous avons : $\Sigma_{l_{[1:1]}} \models \Sigma_{l_{[1:k]}}$, $\Sigma_{l_{[1:2]}} \models \Sigma_{l_{[1:k]}}, \ldots, \Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}, \ldots, \Sigma_{l_{[1:k-2]}} \models \Sigma_{l_{[1:k]}}$ et $\Sigma_{l_{[1:k-1]}} \models \Sigma_{l_{[1:k]}}$. $\Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}$ contredit notre hypothèse de départ, par conséquent $\Sigma \not\models \Sigma_{l_{[1:k]}}$ (CQFD).

Le corollaire est démontré en réitérant cette démonstration pour $\Sigma_{l_{[1:1]}} \not\models \Sigma_{l_{[1:k]}}$, puis $\Sigma_{l_{[1:2]}} \not\models \Sigma_{l_{[1:k]}}, \ldots,$ puis $\Sigma_{l_{[1:j-2]}} \not\models \Sigma_{l_{[1:k]}}$ et enfin $\Sigma_{l_{[1:j-1]}} \not\models \Sigma_{l_{[1:k]}}$. □

Le théorème de partition du modèle « classique » englobe la simplification par les littéraux purs (cf. remarque 7.3). Comme toute application du théorème de partition du modèle « classique » est une application du théorème de partition du modèle généralisé (cf. propriété 7.8), ce dernier permet donc a fortiori d'englober la simplification par les littéraux purs. Par ailleurs, nous montrons que la simplification par les littéraux unitaires est également un cas particulier du théorème de partition du modèle généralisé.

Propriété 7.10
Si un ensemble de clauses Σ *possède un littéral unitaire* l, *alors* $\Sigma \models \Sigma_l$.

Preuve (Propriété 7.10)
Il est possible de distinguer deux types de clauses dans Σ_l :

1. des clauses identiques à des clauses appartenant à Σ : ces clauses de Σ ne contiennent ni l ni $\sim l$ et apparaissent donc dans Σ_l. Il est clair que tout modèle M de Σ satisfait ces clauses de Σ_l puisque M satisfait les mêmes clauses dans Σ ;
2. des clauses issues de clauses de Σ telles que ces clauses de Σ contiennent $\sim l$. Les clauses de Σ_l issues de la suppresion des occurrences de $\sim l$ sont satisfaites par tout modèle de Σ. En effet, comme l est un littéral unitaire tout modèle M de Σ est tel que $M(l) = \mathbb{V}$ et satisfait donc les sous-clauses extraites de Σ par la suppression de $\sim l$.

Donc tout modèle de Σ satisfait toutes les clauses de Σ_l, ainsi tout modèle de Σ satisfait Σ_l et par conséquent $\Sigma \models \Sigma_l$. □

Par extension, nous avons :

Corollaire 7.2
Soit Σ un ensemble de clauses, alors $\Sigma \models \Sigma^*$.

Preuve (Corollaire 7.2)
Soit $\{l_1, l_2, \ldots, l_n\}$ l'ensemble des littéraux unitaires intervenant dans la propagation unitaire de Σ. Nous avons, d'après la propriété 7.10 :
$\Sigma \models \Sigma_{l_1}$, $\Sigma_{l_1} \models \Sigma_{\{l_1,l_2\}}$, ... , $\Sigma_{\{l_1,l_2,\ldots,l_{n-2},l_{n-1}\}} \models \Sigma_{\{l_1,l_2,\ldots,l_{n-1}\}}$ et $\Sigma_{\{l_1,l_2,\ldots,l_{n-1}\}} \models \Sigma^*$.
Par transitivité de la déduction logique ($\models$), on déduit que $\Sigma \models \Sigma^*$. □

Le corollaire 7.2 montre que les branches coupées par l'application du théorème de partition du modèle incluent celles coupées par la propagation unitaire, plus généralement celles coupées par la simplification des littéraux impliqués.

Propriété 7.11
Soient Σ un ensemble de clauses et l un littéral. Si $\Sigma \models l$, alors $\Sigma \models \Sigma_l$.

Preuve (Propriété 7.11)
Si $\Sigma \models l$, alors tout modèle de Σ satisfait le littéral l, donc tout modèle M de Σ est tel que $M(l) = \mathbb{V}$. De cette constatation et en suivant une démarche identique à celle utilisée dans la preuve de la propriété 7.10, nous obtenons $\Sigma \models \Sigma_l$. □

Par ailleurs, l'introduction du théorème de partition du modèle généralisé au sein de l'algorithme de Davis & Putnam peut se faire de la même manière que pour le théorème de partition du modèle « classique ». Cependant détecter une inclusion entre deux ensembles de clauses est une opération rapide et qui s'effectue en temps linéaire avec une structure de données adaptée (cf. remarque 7.3). Il n'en est pas de même pour la conséquence logique. En effet, tester si un ensemble de clauses est une conséquence sémantique d'un autre ensemble de clauses est un problème CoNP-complet. Cette opération nécessite donc le plus souvent un traitement en temps exponentiel par rapport à la taille des ensembles de clauses. Ainsi une application directe au sein de DP du théorème de partition du modèle généralisé, afin de permettre une résolution plus efficace du problème SAT, n'est pas envisageable en pratique. De cette constatation, nous proposons dans le paragraphe suivant d'affaiblir la relation (la conséquence sémantique) entre les deux ensembles de clauses afin de permettre une détection du théorème de partition du modèle généralisé (restreint) en temps polynomial.

En synthèse de ce paragraphe, nous avons :

- proposé une généralisation du théorème de partition du modèle ;
- montré l'incrémentalité du théorème de partition du modèle généralisé (cf. propriété 7.9 et corollaire 7.1) ;
- montré que le théorème de partition du modèle généralisé englobe :
  1. le théorème de partition du modèle « classique » ;
  2. la simplification par les littéraux purs ;
  3. la simplification par les littéraux unitaires ;
  4. et plus généralement la simplification par les littéraux impliqués ;
- constaté que seule une application resteinte du théorème de partition du modèle est envisageable en pratique.

7.2.3 Théorème de partition du modèle généralisé restreint à la propagation unitaire

Afin de rendre possible l'exploitation pratique du théorème de partition du modèle généralisé, il est nécessaire d'affaiblir la relation de déduction utilisée.

Nous proposons de remplacer la conséquence logique ($\models$) par la conséquence logique restreinte à la propagation unitaire ($\models^*$).

7.2.3.1 Définitions et propriétés fondamentales

Définition 7.3 (conséquence logique restreinte à la propagation unitaire)
Soient Σ et Ω deux ensembles de clauses, Ω est une **conséquence logique restreinte à la propagation unitaire** *de Σ , notée $\Sigma \models^* \Omega$, si et seulement si pour toute clause c de Ω, $(\Sigma \wedge \neg c)^*$ contient la clause vide.*

Trivialement, nous avons :

Propriété 7.12
Si $\Sigma \models^* \Omega$ alors $\Sigma \models \Omega$.

Preuve (Propriété 7.12)
Si $\Sigma \models^* \Omega$, alors pour toute clause c de Ω, $(\Sigma \wedge \neg c)^*$ et inconsistant, donc $(\Sigma \wedge \neg\Omega)$ est inconsistant. Par conséquent $\Sigma \models \Omega$. □

Ce qui nous permet d'obtenir le théorème suivant :

Théorème 7.3 (Partition du modèle généralisé restreint à la propagation unitaire)
Soient Σ un ensemble de clauses et $\{l_1, \ldots, l_j, \ldots, l_k\}$ un ensemble fondamental de littéraux.
Si $\Sigma_{l_{[1:j]}} \models^* \Sigma_{l_{[1:k]}} (j \leqslant k)$, alors $\Sigma_{l_{[1:j]}}$ est satisfaisable si et seulement si $\Sigma_{l_{[1:k]}}$ est satisfaisable.

Preuve (Théorème 7.3)
$\Sigma_{l_{[1:j]}} \models^* \Sigma_{l_{[1:k]}} \Rightarrow \Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}$ (propriété 7.12) et si $\Sigma_{l_{[1:j]}} \models \Sigma_{l_{[1:k]}}$ alors $\Sigma_{l_{[1:j]}}$ est satisfaisable si et seulement si $\Sigma_{l_{[1:k]}}$ est satisfaisable (théorème 7.2). □

Nous avons montré dans le paragraphe 7.2.2 que l'application du théorème de partition du modèle généralisé permet entre autres d'englober la simplification par les littéraux purs, unitaires et plus généralement tous les littéraux impliqués. Par ailleurs la restriction de ce théorème à l'inclusion ensembliste ne permet plus de garantir la détection des littéraux unitaires et impliqués (seule la simplification par les littéraux purs est englobée dans le théorème de partition du modèle « classique », cf. remarque 7.3 page 110). Nous montrons que le théorème de partition du modèle généralisé restreint à la propagation unitaire généralise le théorème de partition du modèle généralisé restreint à l'inclusion ensembliste (que nous appelons également théorème de partition du modèle « classique »).

Propriété 7.13
Soient Σ et Ω deux ensembles de clauses. Si $\Omega \subseteq \Sigma$, alors $\Sigma \models^* \Omega$.

Preuve (Propriété 7.13)
Si $\Omega \subseteq \Sigma$, alors chaque clause c de Ω apparaît dans Σ. Donc $\forall\ c \in \Omega,\ \exists c' \in \Sigma$ telle que $c = c'$, par conséquent $(c' \wedge \neg c)^*$ est réduit à la clause vide. Ansi nous avons pour chaque clause c de Ω, $(\Sigma \wedge \neg c)^*$ contient la clause vide. Par conséquent $\Sigma \models^* \Omega$. □

Remarque 7.5
Il faut noter que la réciproque est fausse, comme l'illustre l'exemple ci-dessous.

Exemple 7.7 (Exemple de non réciprocité de la propriété 7.13)
Soient $\Sigma = \{(a \vee b \vee c), (a \vee \neg b), (b \vee \neg c), (c \vee \neg a)\}$ et $\Omega = \Sigma_b = \{(a), (c \vee \neg a)\}$, nous avons bien $\Omega \models^* \Sigma$ mais $\Omega \not\subset \Sigma$.

Ainsi, la propriété ci-dessus nous assure que toute application du théorème de partition du modèle « classique » est une application du théorème de partition du modèle généralisé restreint à la propagation unitaire. Par conséquent, la simplification des littéraux purs est implicitement contenue dans le théorème de partition du modèle généralisé restreint à la propagation unitaire. Nous montrons également que ce théorème effectue également la simplification par les littéraux unitaires.

Propriété 7.14
Pour tout ensemble de clauses Σ : $\Sigma \models^* \Sigma^*$.

Preuve (Propriété 7.14)
Triviale d'après les définitions de Σ^* et de la relation $\models^*$.

□

Au travers de cette section, nous avons montré que le théorème de partition de modèle généralisé restreint à la propagation unitaire permet également de récupérer toutes les simplifications dues :

1° au théorème de partition du modèle « classique » ;

2° aux littéraux purs ;
3° aux littéraux unitaires.

Comparativement au théorème de partition du modèle généralisé, seule la simplification par les littéraux impliqués n'est plus incluse dans le théorème de partition du modèle généralisé restreint à la propagation unitaire, comme l'illustre l'exemple ci-dessous. Cependant la non-inclusion de cette simplification permet de détecter en temps polynomial toute application du théorème de partition du modèle généralisé restreint à la propagation unitaire. Nous présentons cette technique de détection dans le prochain paragraphe.

Exemple 7.8 (Exemple de la non détection des littéraux impliqués)
Soit l'ensemble de clauses $\Sigma = \{(a \vee b \vee c), (a \vee b \vee \neg c), (a \vee \neg b \vee \neg c), (\neg a \vee b \vee \neg c), (\neg a \vee \neg b \vee c), (a \vee \neg b \vee c)\}$. Nous avons $\Sigma \vDash a$ mais $\Sigma \nvDash^* a$.

7.2.3.2 Propriétés pratiques et implantation

L'étude réalisée sur les méthodes de simplification pour l'algorithme de Davis & Putnam (cf. paragraphe 7.1.1) montre l'importance de la capacité à détecter efficacement que telle ou telle méthode de simplification s'applique à un nœud donné de l'arbre de recherche de DP. Par exemple, la simplification de l'ensemble de clauses par les clauses sous-sommées est rarement effectuée car trop fastidieuse et trop prohibitive en temps.

Nous présentons dans cette section des propriétés pratiques du théorème de partition du modèle généralisé restreint à la propagation unitaire permettant de détecter efficacement son application durant l'algorithme de Davis & Putnam. Nous proposons également une implantation de DP intégrant le théorème de partition du modèle généralisé restreint à la propagation unitaire.

La première propriété pratique porte sur la polynomialité du test de la conséquence logique restreinte à la propagation unitaire.

Propriété 7.15
Soient Σ et Ω deux ensembles de clauses. Vérifier si $\Sigma \vDash^* \Omega$ est une opération polynomiale en temps et plus précisément en $\mathcal{O}(C_\Omega \times (|\Sigma| + |c|))$ où C_Ω est le nombre de clauses de Ω et c la plus longue clause de Ω.

Preuve (Propriété 7.15)
Trivialement, par définition de la conséquence logique, C_Ω preuves par propagation unitaire sont nécessaires pour prouver que $\Sigma \vDash^* \Omega$, d'où $\mathcal{O}(C_\Omega \times f(\Sigma \wedge \neg c))$, où $f(\Sigma \wedge \neg c)$ est la complexité de la propagation unitaire de l'ensemble de clauses $(\Sigma \wedge \neg c)$. Par ailleurs la propagation unitaire requiert en temps linéaire $(\mathcal{O}(|\Sigma| + |c|))$ par rapport à la taille de l'instance considérée [Dowling & Gallier 1984]. En conséquence, le test $\Sigma \vDash^* \Omega$ est une opération s'effectuant en $\mathcal{O}(C_\Omega \times (|\Sigma| + |c|))$. □

Polynomialité n'étant malheureusement pas toujours synonyme d'efficacité pratique, nous montrons de quelle manière il est possible de réduire d'un facteur non négligeable le nombre de propagations unitaires nécessaires à la détection du théorème de partition du modèle généralisé restreint à la propagation unitaire lors de l'exécution de DP.

Propriété 7.16
Soient Σ un ensemble de clauses et l un littéral, $\Sigma \vDash^* \Sigma_l$ si et seulement si pour chaque clause c de Σ_l telle

que c provient d'une clause (de Σ) contenant $\sim l$, $(\Sigma \wedge \neg c)^*$ contient la clause vide.

Preuve (Propriété 7.16)
$\Sigma \models^* \Sigma_l$ si et seulement si pour chaque clause c de Σ_l, $(\Sigma \neg c)^*$ contient la clause vide. Or Σ_l est constitué de deux types de clauses : des clauses identiques à des clauses de Σ et des clauses extraites de Σ par la suppression des occurrences de $\sim l$. Nous montrons que pour la première catégorie de clauses, vérifier si $(\Sigma \wedge \neg c)^*$ contient la clause vide est inutile. En effet, c appartient à Σ, donc $(c \wedge \neg c)^*$ conduit obligatoirement à la clause vide. Par conséquent, seule la seconde catégorie de clauses doit être testée (CQFD). □

Des propriétés 7.15 et 7.16, nous déduisons le corollaire suivant :

Corollaire 7.3
Soient Σ un ensemble de clauses et l un littéral de Σ. Vérifier si $\Sigma \models^* \Sigma_l$ est une opération nécessitant un temps en $\mathcal{O}(Occ(\sim l) \times C_\Sigma)$ où $Occ(\sim l)$ représente le nombre d'occurrences de $\sim l$ dans Σ.

Preuve (Corollaire 7.3)
Conséquence immédiate des propriétés 7.15 et 7.16. □

Ce dernier corollaire montre que détecter une application du théorème de partition du modèle généralisé restreint à la propagation unitaire, entre deux nœuds consécutifs de l'arbre de l'algorithme de DP, est une opération simple et très peu coûteuse en temps. Cependant, il est posssible que le théorème s'applique également entre deux nœuds non consécutifs. La propriété suivante et plus particulièrement son corollaire permettent d'éviter tout test inutile et permettent par extension d'obtenir un algorithme de détection incrémental de l'application du théorème et ceci entre n'importe quels nœuds de l'arbre de DP (consécutifs ou non).

Propriété 7.17
Soient Σ un ensemble de clauses et $\{l_1, l_2 \ldots, l_i, \ldots, l_k\}$ un ensemble de littéraux. Si $\Sigma_{l_{[1:i]}} \models^* \Sigma_{l_{[1:k]}}$ $(i \leqslant k)$, alors $\Sigma_{l_{[1:i+1]}} \models^* \Sigma_{l_{[1:k]}}$ et $\Sigma_{l_{[1:i+2]}} \models^* \Sigma_{l_{[1:k]}}$ et ... et $\Sigma_{l_{[1:k-1]}} \models^* \Sigma_{l_{[1:k]}}$.

Preuve (Propriété 7.17)
Dans un premier temps, montrons que si $\Sigma_{l_{[1:i]}} \models^* \Sigma_{l_{[1:k]}}$ alors $\Sigma_{l_{[1:i+1]}} \models^* \Sigma_{l_{[1:k]}}$.

Si $\Sigma_{l_{[1:i]}} \models^* \Sigma_{l_{[1:k]}}$ alors pour toute clause c appartenant à $\Sigma_{l_{[1:k]}}$, l'ensemble de clauses $(\Sigma_{l_{[1:i]}} \wedge \neg c)^*$ contient la clause vide. Donc, a fortiori, $(\Sigma_{l_{[1:i]}} \wedge \{l_{i+1}\} \wedge \neg c)^*$ contient la clause vide.
Par ailleurs $\Sigma_{l_{[1:i+1]}} \equiv (\Sigma_{l_{[1:i]}} \wedge \{l_{i+1}\})$. Par conséquent, $\forall c \in \Sigma_{l_{[1:k]}}$, $(\Sigma_{l_{[1:i+1]}} \wedge \neg c)^*$ contient la clause vide, ce qui signifie, par définition, que $\Sigma_{l_{[1:i+1]}} \models^* \Sigma_{l_{[1:k]}}$.

Un raisonnement identique est applicable pour $\Sigma_{l_{[1:i+2]}}$, $\Sigma_{l_{[1:i+3]}}$, ... , $\Sigma_{l_{[1:k-1]}}$. □

Corollaire 7.4
Soient Σ un ensemble de clauses et $\{l_1, l_2 \ldots, l_i, \ldots, l_k\}$ un ensemble de littéraux. Si $\Sigma_{l_{[1:i]}} \not\models^* \Sigma_{l_{[1:k]}}$ $(i \leqslant k)$, alors $\Sigma_{l_{[1:i-1]}} \not\models^* \Sigma_{l_{[1:k]}}$ et $\Sigma_{l_{[1:i-2]}} \not\models^* \Sigma_{l_{[1:k]}}$ et ... et $\Sigma \not\models^* \Sigma_{l_{[1:k]}}$.

Preuve (Corollaire 7.4)
Dans un premier temps, montrons que si $\Sigma_{l_{[1:i]}} \not\models^* \Sigma_{l_{[1:k]}}$ alors $\Sigma_{l_{[1:i-1]}} \not\models^* \Sigma_{l_{[1:k]}}$.

Nous avons $\Sigma_{l_{[1:i]}} \not\models^* \Sigma_{l_{[1:k]}}$ et supposons que $\Sigma_{l_{[1:i-1]}} \models^* \Sigma_{l_{[1:k]}}$. Si $\Sigma_{l_{[1:i-1]}} \models^* \Sigma_{l_{[1:k]}}$, alors $\Sigma_{l_{[1:i]}} \models^* \Sigma_{l_{[1:k]}}$ d'après la propriété 7.17. Cette conclusion est contradictoire avec l'hypothèse, par conséquent nous

avons démontré par l'absurde que $\Sigma_{l_{[1:i-1]}} \not\models^* \Sigma_{l_{[1:k]}}$.

Un raisonnement identique est applicable pour $\Sigma_{l_{[1:i-2]}}$, $\Sigma_{l_{[1:i-3]}}$, ... , Σ. □

Ainsi, nous avons prouvé que l'application du théorème de partition du modèle généralisé restreint à la propagation unitaire entre deux nœuds non consécutifs de l'arbre de DP est assujettie à l'application du théorème entre deux nœuds consécutifs. À partir de cette constatation, nous proposons une implantation du théorème de partition du modèle généralisé restreint à la propagation unitaire.

L'algorithme proposé cherche à appliquer le théorème lors de la « remontée » dans l'arbre de DP. Ce type d'implantation a été préféré à une implantation cherchant à appliquer le théorème dans la « descente », uniquement pour des raisons évidentes d'efficacité. En effet, cette technique permet notamment d'éviter des propagations unitaires inutiles si la branche développée est satisfaisable.

Les algorithmes suivants décrivent la procédure DP intégrant la détection du théorème (cf. algorithme 7.5) et la procédure de détection proprement dite (cf. algorithme 7.6).

Algorithme 7.5 DP ET PARTITION DU MODÈLE GÉNÉRALISÉ RESTREINT À LA PROPAG. UNIT.

```
1.  Function DP_PMGRPU : boolean
2.  Input  : un ensemble de clauses Σ
3.  Output : true si Σ est consistant, false sinon
4.  Begin
5.     Σ*=Propagation_Unitaire(Σ) ;
6.     if (Σ* contient la clause vide) then return false ;
7.     elif (Σ*==∅) then return true ;
8.     else
9.       l=Heuristique_de_branchement(Σ*) ;
10.      if (DP_PMGRPU(Σ* ∪ {(l)})) then return true ;
11.      else return Part_Modele_Gen_Rest_Propag_Unit(Σ*, l) ;
12.      fi
13.    fi
14. End
```

Lors de l'appel par DP à `Part_Modele_Gen_Rest_Propag_Unit`, la branche Σ_l a été explorée et prouvée inconsistante. Si $\Sigma \models^* \Sigma_l$ alors la branche $\Sigma_{\sim l}$ peut être coupée d'après le théorème de partition du modèle généralisé restreint à la propagation unitaire. L'algorithme ci-dessous se charge d'effectuer ce test et de couper ou d'explorer la branche suivant le résultat du test.

Algorithme 7.6 PARTITION DU MODÈLE GÉNÉRALISÉ RESTREINT À LA PROPAGATION UNITAIRE

```
1.  Function Part_Modele_Gen_Rest_Propag_Unit : boolean
2.  Input  : un ensemble de clauses Σ et un littéral l tel que
             Σ_l a été prouvé inconsistant ;
3.  Output : true si Σ_~l est consistant, false sinon ;
4.  Begin
5.    Σ*_~l = Propagation_Unitaire(Σ ∪ {~l}) ;
6.    if (Σ*_~l contient la clause vide) then return false ;
7.    elif (Σ*_~l==∅) then return true ;
```

```
8.    else                    %% Tester Σ ⊨* Σ_l revient à vérifier pour chaque clause c de Σ
                                                  %% telle que c = {∼l ∨ c'}, si (Σ ⊨* c')
9.      application = true ;     %% Variable booléenne true tant que le théo. s'applique
10.     foreach clause c de Σ telle que c = {∼l ∨ l_1 ∨ l_2 ∨ ... ∨ l_n}
11.     do
12.       if (Propagation_Unitaire(Σ*_∼l ∪ {∼l_1} ∪ ... ∪ {∼l_n}) contient la clause vide)
13.       then Σ*_∼l = (Σ*_∼l ∪ {l_1 ∨ l_2 ∨ ... ∨ l_n}) ;
14.       else application = false ;
15.     done
16.     if (application) then return false ;          %% application ⇒ branche coupée
17.     else return DP_PMGRPU(Σ*_∼l) ;              %% non application ⇒ branche explorée
18.     fi
19.   fi
20. End
```

Nous terminons ce paragraphe par deux remarques concernant l'algorithme de détection du théorème de partition du modèle généralisé restreint à la propagation unitaire (cf. algorithme 7.6 page précédente).

1° L'algorithme débute par la propagation unitaire de $\sim l$. Nous avons décidé de commencer la procédure par cette propagation pour des raisons d'efficacité. En effet, pour chacune des clauses à tester lors de la vérification de $\Sigma \models^* \Sigma_l$, il est clair que le mono-littéral $\sim l$ est produit. Par conséquent afin d'éviter de répéter n[4] fois la production du mono-littéral $\sim l$ et les éventuelles propagations unitaires engendrées par ce mono-littéral, la propagation unitaire de $\sim l$ est effectuée une fois pour toutes en début de procédure.

2° Lorsqu'une clause c' est prouvée être une conséquence logique restreinte à la propagation unitaire de $\Sigma^*_{\sim l}$, cette clause est ajoutée à l'ensemble de clauses $\Sigma^*_{\sim l}$. Cet ajout de clauses est réalisé en ligne 13 de l'algorithme 7.6. Il permet non seulement d'accélérer le temps de calcul des différentes propagations unitaires effectuées lors du test d'application du théorème, mais également de réduire la taille du sous-arbre construit par DP lors de l'exploration de $\Sigma_{\sim l}$ lorsque le théorème n'a pu être appliqué.
Il est à noter que cet ajout de clauses se fait en espace constant. En effet, la clause c' ajoutée à $\Sigma^*_{\sim l}$ est extraite d'une clause ($c = (\sim l \vee c')$) appartenant à Σ. Par conséquent, comme cette clause c de Σ a été effacée dans $\Sigma^*_{\sim l}$ par la propagation unitaire de $\sim l$, l'ajout de c' s'effectue donc par une simple « réactivation » d'une sous-clause de Σ. Cette opération est réalisée en temps et en espace constant avec une structure de données adaptée.
Par ailleurs, la version de l'agorithme présentée dans ce manuscrit préconise la production d'un maximum de clauses impliquées. En effet, dès lors qu'une clause n'est pas impliquée, la preuve est faite que $\Sigma \not\models^* \Sigma_l$, par conséquent, afin de minimiser le temps de calcul, il serait judicieux de quitter la boucle prématurément et de lancer immédiatement l'exploration de la branche correspondant à $\Sigma^*_{\sim l}$, ceci sans vérifier si d'autres clauses peuvent être ajoutées. À l'inverse, la version présentée est « jusqu'au-boutiste », elle cherche à impliquer un maximum de clauses, ceci afin de réduire au maximum la taille du sous-arbre non encore exploré. Nous discutons dans le paragraphe dédié aux extensions de l'algorithme, diverses stratégies envisagées quant à l'arrêt plus ou moins prématuré du test d'implication de clauses.

Résultats expérimentaux

Afin de valider notre approche nous avons effectué une série de tests portant sur des instances *3***SAT** aléatoires. Ces tests ont pour objectif de mesurer l'effet du théorème de partition du modèle généralisé

4. n représente ici le nombre de clauses à tester, c'est-à-dire le nombre d'occurrences de $\sim l$ dans Σ.

restreint à la propagation unitaire sur l'arbre de recherche de DP et plus précisément sur le nombre de nœuds de cet arbre. Pour ce faire nous avons comparé notre approche avec l'algorithme de Davis & Putnam standard (sans mécanisme de simplification) et avec l'algorithme d'évaluation sémantique.

Par ailleurs, nous avons également mesuré cet effet à différents rapports nombre de clauses sur nombre de variables afin de déterminer dans quelle zone (sous-contrainte, critique ou sur-contrainte) notre approche obtenait les meilleurs résultats comparativement aux algorithmes précités.

De plus, nous avons testé deux manières distinctes d'appliquer le théorème de partition du modèle généralisé restreint à la propagation unitaire. La première d'entre elles est celle décrite dans la section précédente (cf. algorithme 7.6) et correspond à une vérification « jusqu'au-boutiste » du test d'application du théorème. C'est-à-dire que chaque clause pour laquelle l'implication doit être testée, est traitée, même si l'une d'entre elles a déjà été prouvée non impliquée. À l'inverse la version « arrêt » stoppe la procédure de vérification dès lors qu'une clause n'a pas été impliquée. Ces deux versions sont discutées plus précisément dans la section suivante.

Ces quatre algorithmes ont été expérimentés sur des instances *3***SAT** pour lesquelles le nombre de variable varie entre 25 et 250 et pour lesquelles le rapport du nombre de variables sur le nombre de clauses varie entre 3.00 et 6.00. Par ailleurs pour chacun des couples (nombre de variables, rapport C/V) 500 instances ont été testées. Les courbes suivantes synthétisent les résultats obtenus par chacune des approches comparativement au nombre de nœuds de l'arbre de décision de DP.

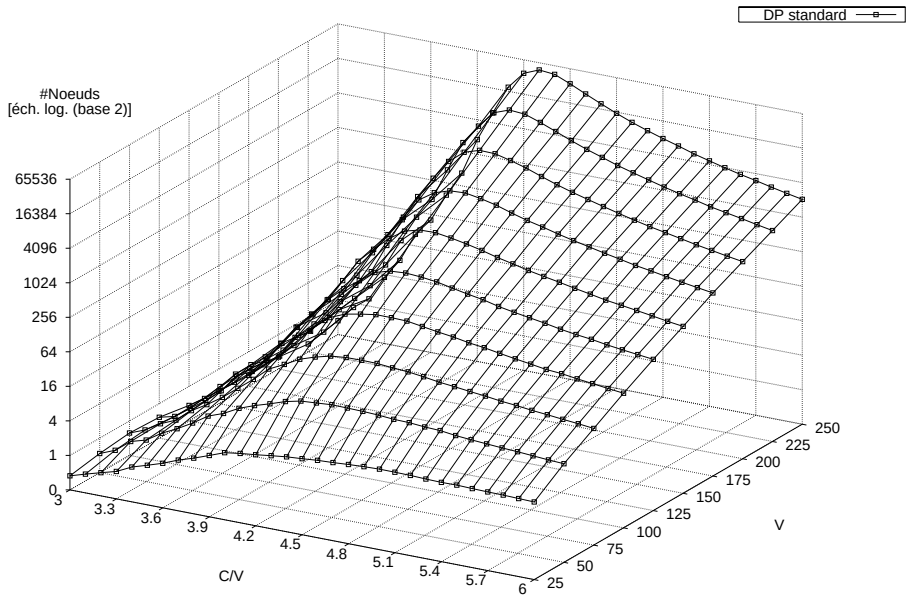

FIG. 7.4 – *Nombre de nœuds de DP « standard » pour des instances 3***SAT**

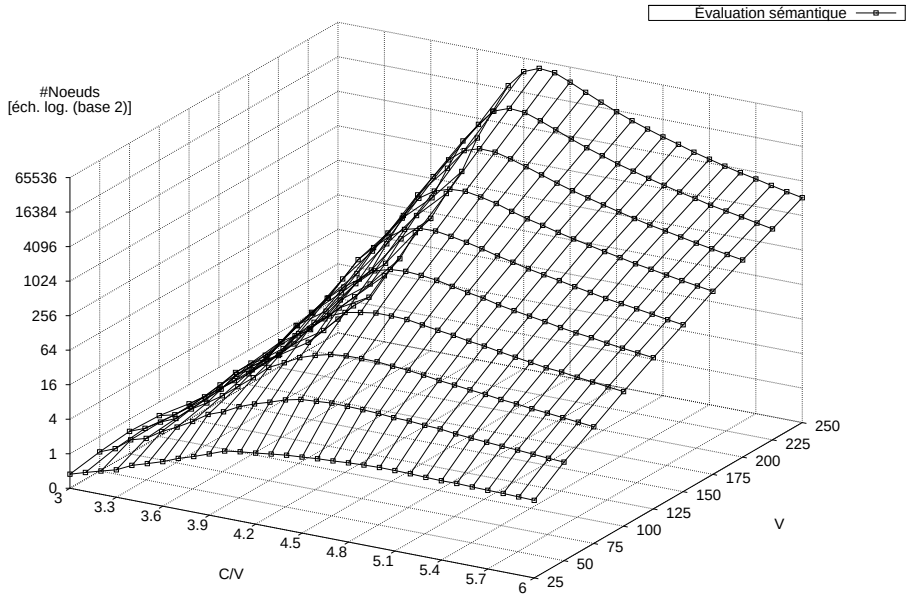

FIG. 7.5 – *Nombre de nœuds de l'évaluation sémantique pour des instances 3***SAT**

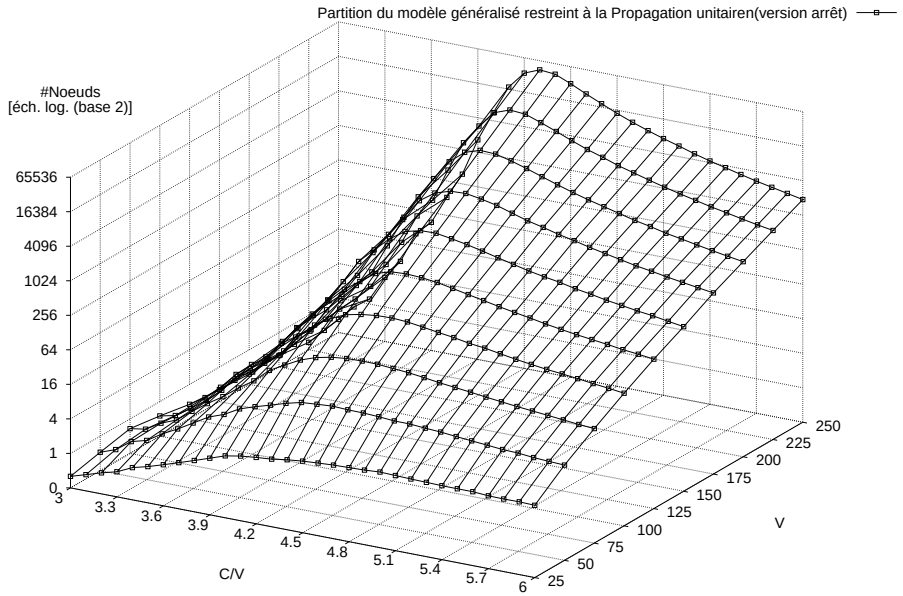

FIG. 7.6 – *Nombre de nœuds de DP utilisant le théorème de partition du modèle généralisé restreint à la propagation unitaire* « *version arrêt* »

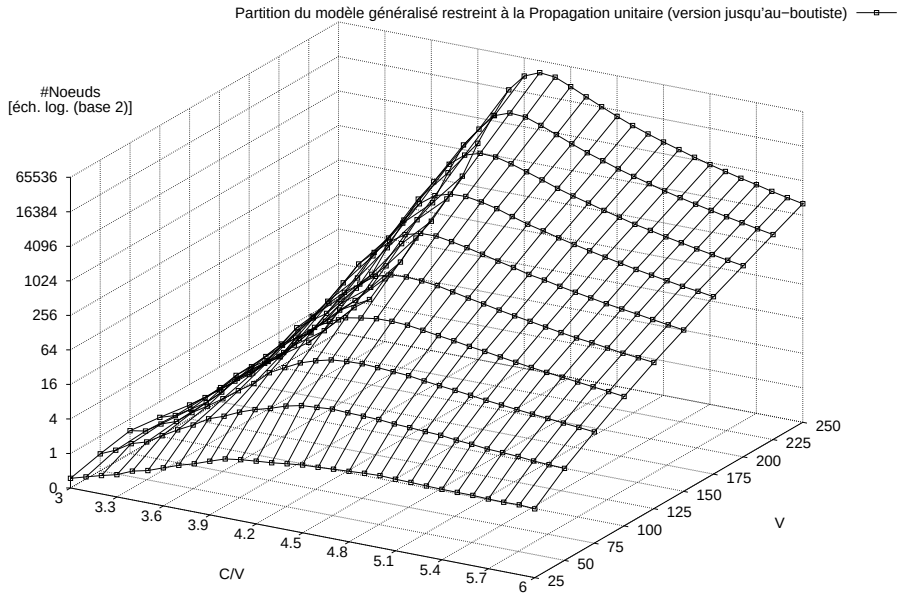

FIG. 7.7 – *Nombre de nœuds de DP utilisant le théorème de partition du modèle généralisé restreint à la propagation unitaire* « *version jusqu'au-boutiste* »

Ces courbes, difficiles de lecture par leur aspect tri-dimensionnel et par l'échelle logarithmique, montrent que quelle que soit la taille du problème, c'est-à-dire quel que soit le couple (nombre de variables, rapport C/V), les approches basées sur le théorème de partition du modèle généralisé restreint à la propagation unitaire admettent toujours moins de nœuds que les procédures de Davis & Putnam standard et d'évaluation sémantique. Par ailleurs, il faut remarquer que sur ces instances aléatoires l'évaluation sémantique n'apporte quasiment rien, c'est-à-dire que le théorème de partition du modèle classique s'applique très rarement (voire pas du tout pour une grande majorité de tailles). La différence entre les approches croît en fonction du nombre de variables. Nous présentons dans la courbe suivante qui regroupe les résultats obtenus pour chaque approche sur les instances de 250 variables.

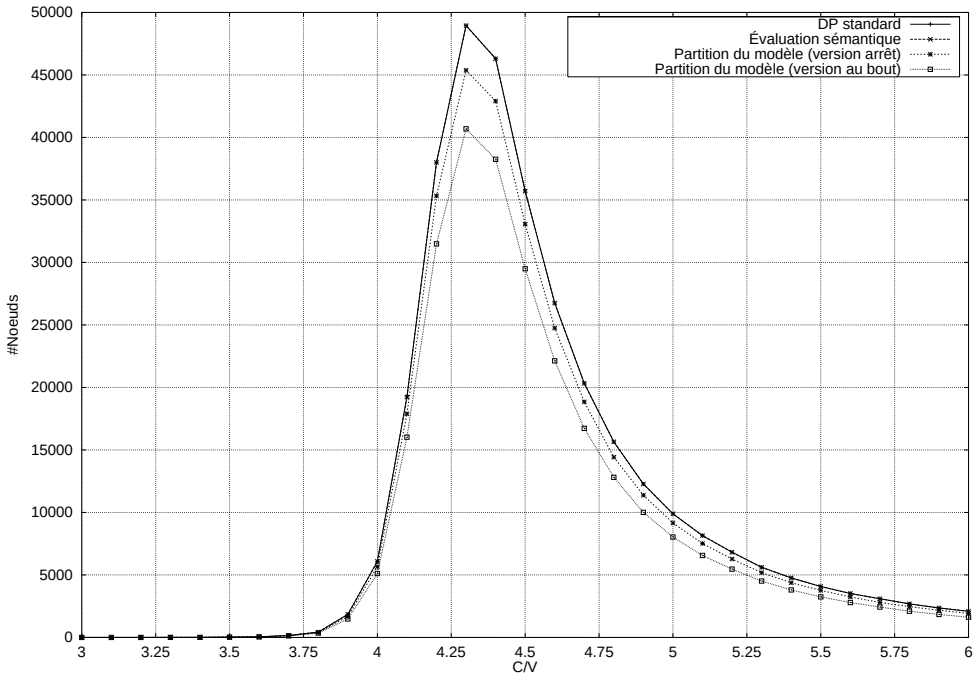

FIG. 7.8 – *Nombre de nœuds en fonction du rapport C/V pour des instances 3***SAT** *aléatoires de* 250 *variables*

Cette courbe fait apparaître très nettement le gain obtenu par les approches basées sur le théorème de partition du modèle généralisé. Il faut également noter que la version « jusqu'au-boutiste » obtient sensiblement de meilleurs résultats que la version « arrêt ». Ceci prouve l'utilité des ajouts de clauses pour le reste de la recherche.

Il reste que si nos approches permettent une diminution notable du nombre de nœuds elles sont d'avantage consommatrices en temps CPU. Les courbes suivantes expriment le temps moyen passé par chacune des méthodes pour résoudre une instance de chaque taille.

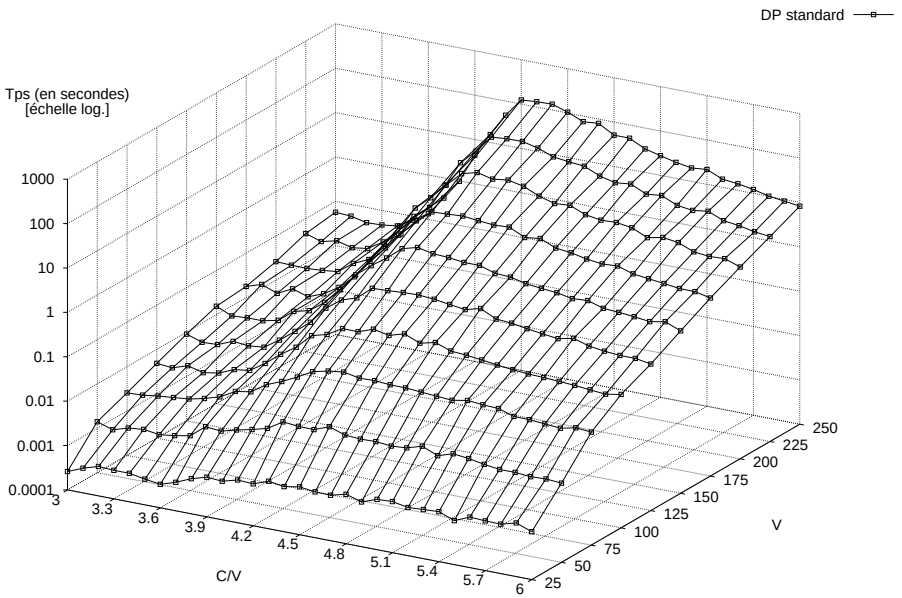

FIG. 7.9 – *Temps moyen de DP « standard » pour résoudre des instances 3***SAT** *aléatoires*

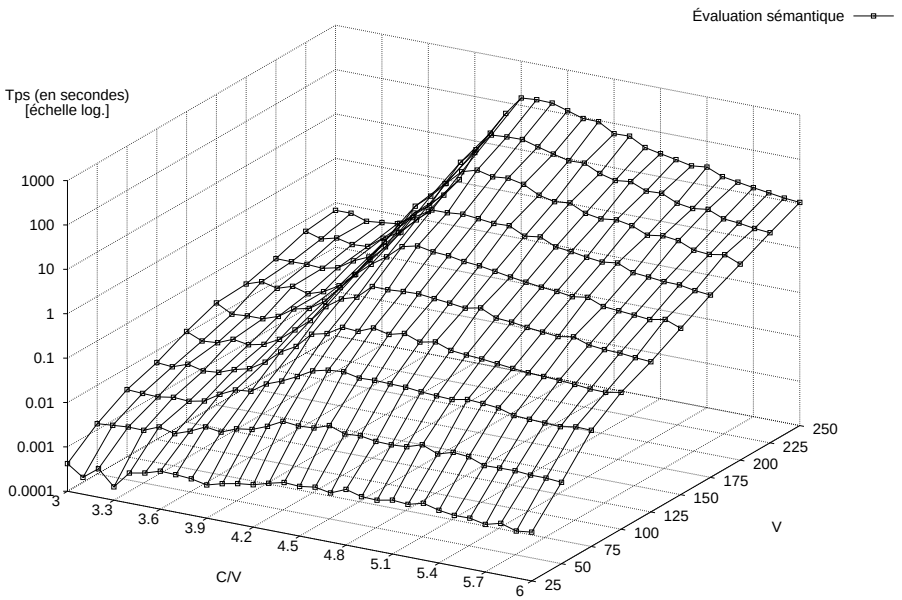

FIG. 7.10 – *Temps moyen de l'évaluation sémantique pour résoudre des instances 3***SAT** *aléatoires*

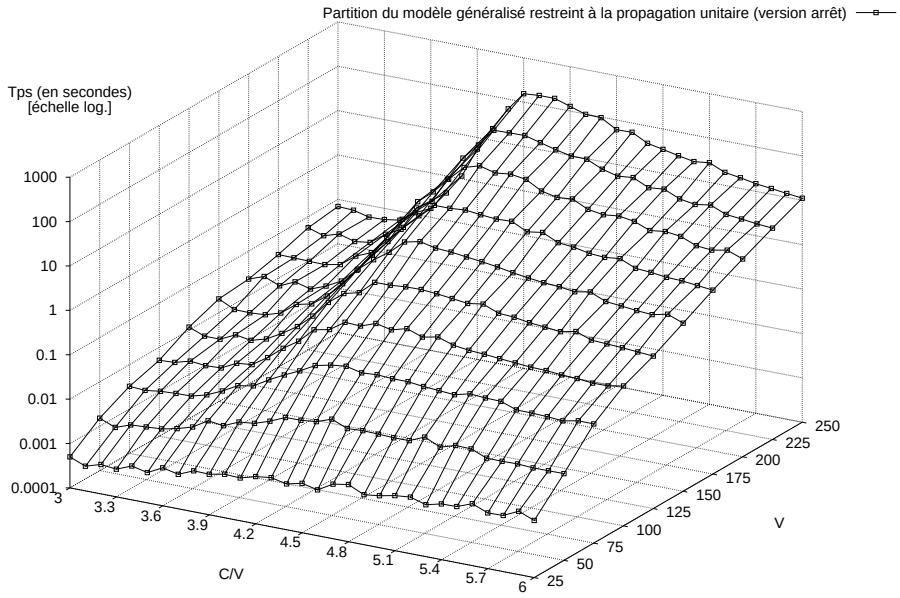

FIG. 7.11 – *Temps moyen de DP utilisant le théorème de partition du modèle généralisé (version « arrêt ») pour résoudre des instances 3***SAT** *aléatoires*

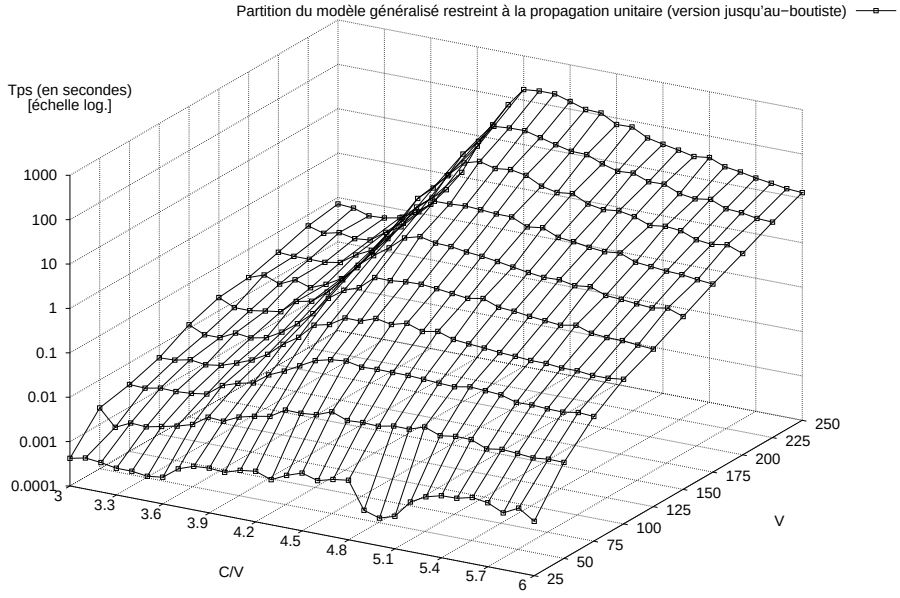

FIG. 7.12 – *Temps moyen de DP utilisant le théorème de partition du modèle généralisé (version « jusqu'au-boutiste ») pour résoudre des instances 3***SAT** *aléatoires*

À l'inverse des courbes précédentes, les approches basées sur le théorème de partition du modèle généralisé semblent plus lentes. La courbe suivante montre les temps obtenus par chacune des méthodes pour un nombre de variable fixe égal à 250.

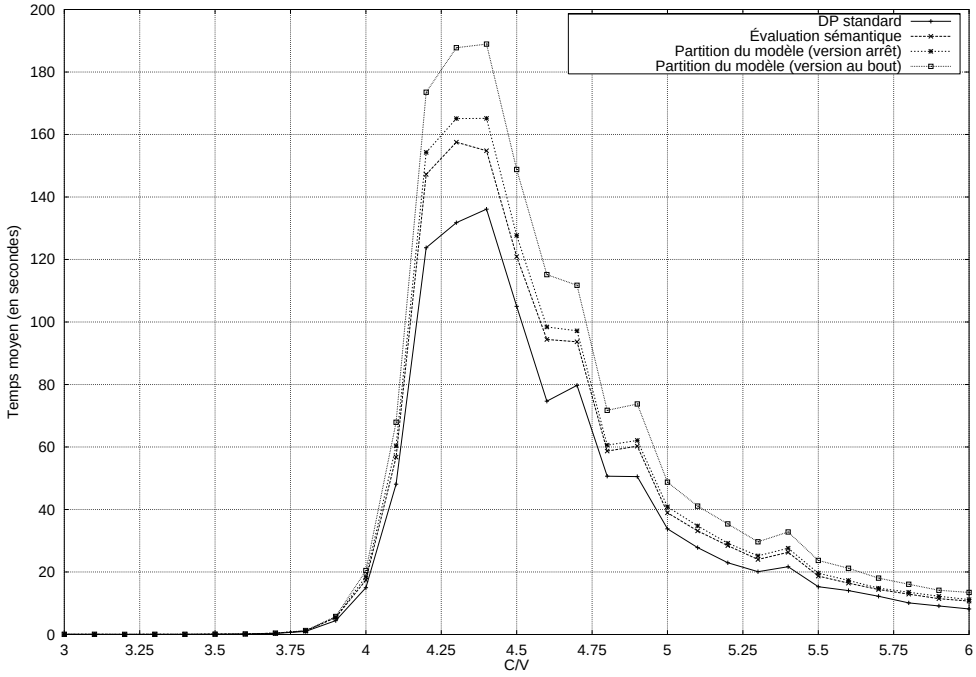

FIG. 7.13 – *Temps d'éxécution en fonction du rapport C/V pour des instances 3***SAT** *aléatoires de* 250 *variables*

Les résultats concernant les temps d'exécution des méthodes basées sur le théorème de partition du modèle généralisé restreint à la propagation unitaire montrent l'importance de trouver des stratégies permettant de réduire au maximum le temps de calcul. Dans la prochaine section nous discutons de diverses extensions dont certaines ont pour objectif la réduction du temps de calcul de ces approches.

Optimisations naturelles

Comme discuté précédemment, deux politiques se font face en ce qui concerne les tests d'implication de clauses. En effet, la version présentée préconise de chercher à impliquer un maximum de clauses, c'est-à-dire de ne pas stopper le test d'application du théorème dès lors que celui-ci a été prouvé impossible à satisfaire. Cette stratégie se justifie par le fait que chaque clause impliquée est ajoutée à la base de connaissances et que cet ajout permet de supprimer des interprétations. Ces suppressions ont pour conséquence de minimiser le nombre de nœuds explorés et de favoriser l'application du théorème dans le reste de la branche. Cependant, bien que la propagation unitaire soit réalisable en temps linéaire, multiplier les tests d'implications peut nuire aux performances globales de l'algorithme. Ainsi, vu que le temps CPU est souvent considéré comme le critère de jugement suprême, une variante à l'algorithme proposé consiste à stopper le test d'application du théorème dès lors que celui est insatisfait. Entre ces deux variantes extrêmes, nous pensons qu'une stratégie intermédiaire peut être imaginée. Celle-ci consisterait à imposer un ordre sur les différents tests d'implications à réaliser. Plus précisément, les clauses qui, selon cet ordre, auraient le plus de chances d'être impliquées seraient testées prioritairement. Ainsi dès lors qu'une clause n'est pas impliquée par Σ, le test d'application du théorème est stoppé en considérant qu'il est peu probable qu'une clause (de rang inférieur selon l'ordre) soit de nouveau impliquée alors que la clause précédente (selon l'ordre) ne l'a pas été. Toute la difficulté d'une telle approche réside dans la détermination de l'ordre. En

effet cet ordre doit pouvoir être calculé de manière efficace, afin de pas nuire aux performances globales de la méthode, et doit être le plus pertinent possible.

Une seconde amélioration réside dans l'application du théorème uniquement à partir d'une certaine profondeur de l'arbre de Davis & Putnam. L'idée est donc de n'appliquer ce théorème que lorsque celui-ci a un maximum de « chances » d'être satisfait. Par exemple, dans le cas d'instances *3***SAT**, les nœuds situés près de la racine de l'arbre de décision de DP ne contiennent que très peu de clauses binaires, en effet il faut un certain nombre d'affectations afin de voir apparaître un nombre significatif de clauses binaires. Ceci est d'autant plus vrai si l'on considère des instances *4***SAT**. Or c'est bien la présence massive de clauses binaires qui favorise la propagation unitaire et donc l'application du théorème. Par ailleurs de nombreuses techniques de simplification basées sur la propagation unitaire, comme celles utilisées dans C-SAT, ne sont appliquées qu'à partir d'une certaine profondeur de l'arbre de recherche de DP. De ces constats, il est donc naturel de ne tester l'application du théorème qu'à partir du moment où un nombre significatif de clauses binaires sont présentes dans Σ. Ce nombre significatif de clauses binaires ne pouvant être atteint le plus souvent qu'à partir d'une profondeur donnée de l'arbre de Davis & Putnam, il est donc naturel de restreindre le test d'application du théorème uniquement lorsqu'une certaine profondeur de l'arbre de DP est atteinte. Il reste à déterminer cette profondeur minimale d'application. Il nous semble que seule une étude empirique sur différentes catégories d'instances (réels, aléatoires, etc.) nous permettra de déterminer la profondeur optimale d'application du théorème.

Nous pensons que les gains obtenus par l'application du théorème de partition du modèle seront d'autant plus importants sur les problèmes structurés. En effet la présence d'une structure dans le problème (souvent matérialisée par de nombreuses clauses binaires) devrait permettre une application accrue du théorème. De plus, la présence de nombreuses résolvantes (courtes), de redondances peut favoriser l'implication de sous-clauses même à la racine de l'arbre. Ce qui nous a amené à nous poser la question suivante : « À partir de quelle niveau de l'arbre de DP une clause est-elle impliquée ? ».

7.2.4 Extension : niveaux d'implications

Les implantations présentées précédemment consistent à tester l'application du théorème à chaque point de choix et permettent de couper uniquement la branche courante. Il serait intéressant de voir dans quelle mesure il est possible de couper plusieurs branches en même temps et d'effectuer ainsi un « *backtracking* » non chronologique, ce qui permet d'éviter de multiples tests d'application du théorème et, par conséquent, un gain de temps substantiel.

Nous montrons dans ce paragraphe qu'il est possible d'atteindre cet objectif en déterminant la partition la plus grande étant donné un ordre de propagation des littéraux unitaires lors du test d'implication d'une clause. Pour ce faire nous introduisons les concepts de « niveau » et de « graphe d'implication ».

Définition 7.4 (Niveau d'un littéral)
Le **niveau** *d'un littéral l, noté $\delta(l)$, est donné par le nombre de branches inexplorées précédant son apparition dans l'interprétation partielle courante.*

Tous les littéraux apparaissant entre deux branches inexplorées admettent donc le même niveau, comme l'illustre l'exemple suivant. Lorsqu'un littéral n'apparaît pas dans l'interprétation courante, son niveau est fixé à l'infini.

Exemple 7.9 (Exemple de niveaux des littéraux pour un arbre de DP)
Soit l'arbre de décision de la figure page ci-contre décrit par la procédure de Davis & Putnam sur une

formule Σ :

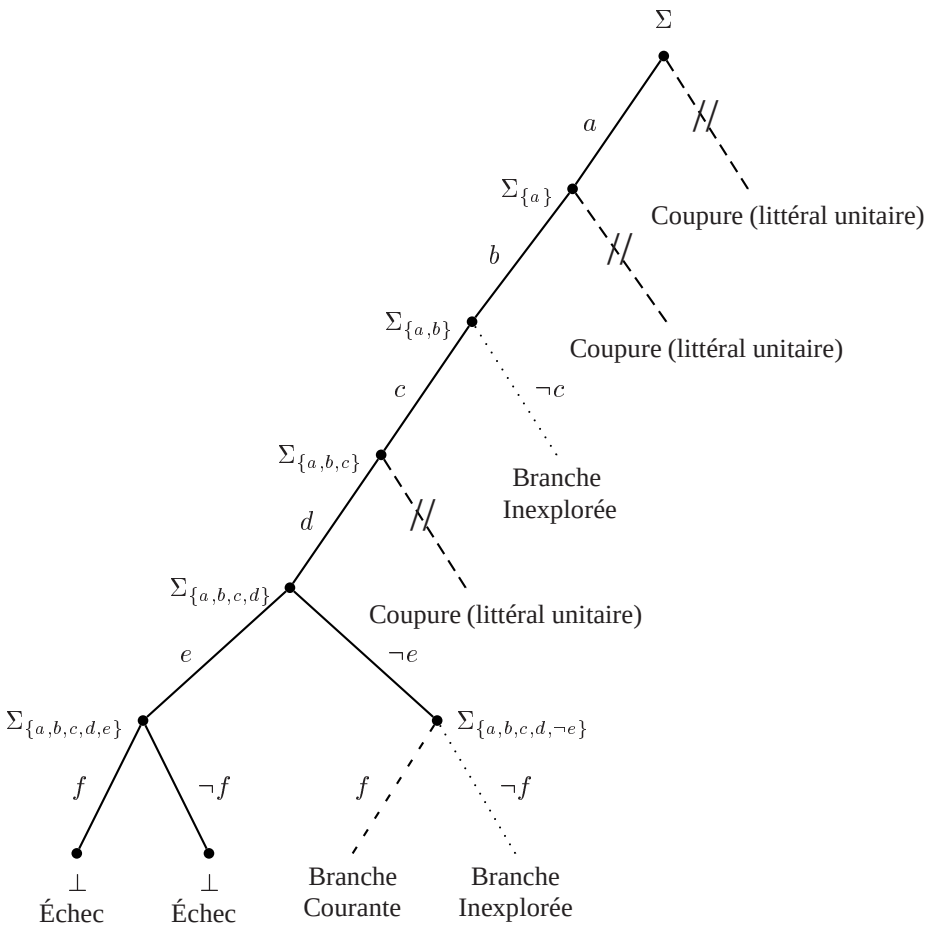

FIG. 7.14 – *Illustration des niveaux des littéraux pour un arbre de recherche de DP*

Pour cet arbre, le tableau suivant décrit le niveau de chacun des littéraux, quel que soit le nœud.

Formule	Niveaux des littéraux											
	$\delta(a)$	$\delta(\neg a)$	$\delta(b)$	$\delta(\neg b)$	$\delta(c)$	$\delta(\neg c)$	$\delta(d)$	$\delta(\neg d)$	$\delta(e)$	$\delta(\neg e)$	$\delta(f)$	$\delta(\neg f)$
Σ	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma_{\{a\}}$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma_{\{a,b\}}$	0	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$\Sigma_{\{a,b,c\}}$	0	∞	0	∞	1	∞	∞	∞	∞	∞	∞	∞
$\Sigma_{\{a,b,\neg c\}}$	0	∞	0	∞	∞	0	∞	∞	∞	∞	∞	∞
$\Sigma_{\{a,b,c,d\}}$	0	∞	0	∞	1	∞	1	∞	∞	∞	∞	∞
$\Sigma_{\{a,b,c,d,e\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	∞	∞
$\Sigma_{\{a,b,c,d,e,f\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	3	∞
$\Sigma_{\{a,b,c,d,e,\neg f\}}$	0	∞	0	∞	1	∞	1	∞	2	∞	∞	2
$\Sigma_{\{a,b,c,d,\neg e\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	∞	∞
$\Sigma_{\{a,b,c,d,\neg e,f\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	2	∞
$\Sigma_{\{a,b,c,d,\neg e,\neg f\}}$	0	∞	0	∞	1	∞	1	∞	∞	1	∞	1

Définition 7.5 (Niveau d'une clause)
Soient Σ un ensemble de clauses et $\{l_1, l_2, \ldots, l_i\}$ un ensemble de littéraux représentant une interprétation partielle de Σ. Si c est une clause de $\Sigma_{l_{[1:i]}}$, alors le **niveau** *de c, notée $\delta(c)$, se définit de la manière suivante :*

- *si $c \in \Sigma$, $\delta(c) = 0$;*
- *sinon $\delta(c) = \max\{\delta(l)$ t.q. $l \in c$ et $\sim l \in \{l_1, l_2, \ldots, l_i\}\}$*

Le niveau d'une clause correspond donc au niveau du dernier littéral ayant raccourci cette clause lors de son assignation.

L'algorithme de Davis & Putnam est modifié de manière à prendre en compte le calcul de ces niveaux. C'est-à-dire qu'initialement chaque littéral admet un niveau infini, lorsqu'une variable est affectée le niveau du littéral correspondant est mis à jour, ainsi que celui des clauses raccourcies par l'affectation de cette variable.

Comme décrit précédemment, l'opération fondamentale du test d'application du théorème de partition du modèle généralisé restreint à la propagation unitaire consiste à déterminer si une clause est impliquée par un ensemble de clauses en utilisant uniquement la propagation unitaire. La séquence d'implications générées par la propagation unitaire est capturée par un graphe orienté acyclique, appelé graphe d'implication. La notion de graphe d'implications a également été utilisée dans GRASP par J.P.M. Silva et K.A. Sakallah pour déterminer les causes de conflits [Silva & Sakallah 1996b].

Définition 7.6 (Graphe d'implication)
Un graphe d'implication, noté $\mathcal{G} = (\mathcal{S}, \mathcal{A})$:

1. *$\mathcal{S}$: chaque sommet, noté $s(l)$, de $\mathcal{G}$ correspond à un littéral l affecté lors de la propagation ;*
2. *$\mathcal{A}$: soit une clause $c = \{\neg l_1, \ldots, \neg l_{i-1}, l_i, \neg l_{i+1}, \ldots, \neg l_n\}$ responsable de l'implication du littéral l_i, c'est-à-dire que si I est l'interprétation partielle courante alors :*
   $$c \cap I = \{l_i\} \text{ et } (c \cap \sim I) = \{l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n\}.$$
   L'ensemble des sommets prédécesseurs du sommet $s(l_i)$ est égal à :
   $$\{s(l_1), \ldots, s(l_{i-1}), s(l_{i+1}), \ldots, s(l_n)\}.$$
   Les arcs $\langle s(l_j), s(l_i)\rangle$ tels que $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$ sont étiquetés par la clause c.
   L'étiquette d'un arc $\langle i, j\rangle$ est notée $val(\langle i, j\rangle) = c$.

Remarque 7.6
Le graphe d'implication d'une clause impliquée par la propagation unitaire admet comme source les sommets étiquetés par la négation de chacun des littéraux de la clause. Par ailleurs, il est évident que ce graphe n'admet pas de cycle (par construction).

Exemple 7.10 (Exemple de graphe d'implication d'une clause)
Soit $\Sigma = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}, c_{14}\}$, avec :

$$\begin{array}{lcl|lcl} c_1 & : & (\neg a \vee c) & c_8 & : & (\neg e \vee h) \\ c_2 & : & (\neg a \vee d) & c_9 & : & (\neg g \vee h \vee \neg f) \\ c_3 & : & (\neg b \vee d) & c_{10} & : & (\neg h \vee k) \\ c_4 & : & (\neg d \vee f) & c_{11} & : & (\neg i \vee l \vee \neg k) \\ c_5 & : & (\neg d \vee g) & c_{12} & : & (\neg i \vee \neg l) \\ c_6 & : & (\neg b \vee e) & c_{13} & : & (\neg k \vee \neg l) \\ c_7 & : & (\neg e \vee i) & c_{14} & : & (c \vee e) \end{array}$$

Le graphe d'implication de la clause $c = \{\neg a \vee \neg b\}$ est représenté dans la figure page suivante.

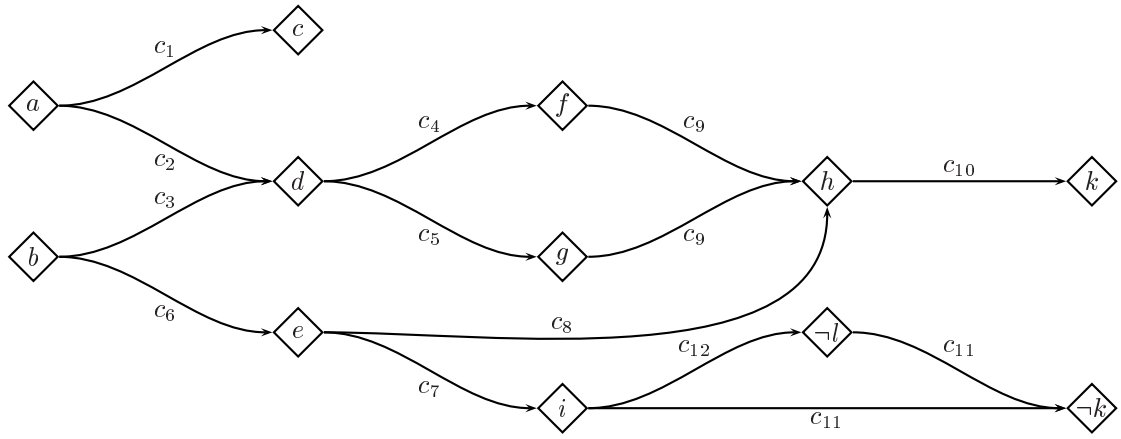

FIG. 7.15 – *Graphe d'implication d'une clause*

Nous décrivons ci-dessous l'algorithme de construction d'un graphe d'implication d'une clause c à partir d'un ensemble de clauses Σ.

Algorithme 7.7 CONSTRUCTION GRAPHE D'IMPLICATION

```
1.  Function CONSTRUCTION_GRAPHE_IMPLICATION : Graphe
2.  Input  : Une liste de littéraux L tel que L = {l|¬l ∈ c}
         et un ensemble de clauses Σ ;
3.  Output : Un graphe G correspondant au graphe d'implication de c par Σ ;
4.  Begin
5.    I = ∅ ;
6.    while (L ≠ ∅)
7.    do
8.       l=le premier élément de L ;   %% L contient la liste des littéraux à propager
9.       L=L \ {l} ;
10.      I = I ∪ {l} ;                 %% I représente l'interprétation partielle courante
11.      foreach clause c de Σ t.q. ¬l ∈ c, c ∩ I = ∅ et (c \ (c ∩ ∼ I)) = {l'}
12.      do
13.        foreach littéral l'' ∈ (∼ c ∩ I)
14.        do
15.           Créer_Arc(G,l'',l',c) ;        %% Ajoute au graphe G un arc orienté
                    %% entre le sommet s(l'') et le sommet s(l') étiqueté par la clause c
16.        done
17.        L=L ∪ {l'} ;                    %% l' est un littéral unitaire à propager
18.      done
19.      return G ;
20. End
```

Afin de déterminer le niveau minimal d'implication de la clause, nous associons à chaque sommet $s(l)$ une valeur, notée $\eta(l)$ se calculant comme suit :

– une valeur nulle est associée aux sources du graphe ;

– soient un sommet $s(l)$ et $\{s(l_{1,1}), \ldots, s(l_{1,m}), s(l_{2,1}), \ldots, s(l_{2,n}), \ldots, s(l_{k,1}), \ldots, s(l_{k,p})\}$ l'ensemble des prédécesseurs de l tel que les arcs $\langle s(l_{1,1}), s(l)\rangle, \ldots, \langle s(l_{1,m}), s(l)\rangle$ sont étiquetés par c_1, les arcs $\langle s(l_{2,1}), s(l)\rangle, \ldots, \langle s(l_{2,n}), s(l)\rangle$ sont étiquetés par c_2, etc.
$\eta(s(l)) = \min[\max(\delta(c_1), \eta(s(l_{1,1})), \ldots, \eta(s(l_{1,m}))), \ldots, \max(\delta(c_k), \eta(s(l_{k,1})), \ldots, \eta(s(l_{k,p})))]$.

Exemple 7.11 (Graphe et niveau d'implication d'une clause)
Si l'on considère le graphe proposé dans l'exemple 7.10 et les niveaux de clauses suivants :

$$\begin{array}{l|l|l|l|l|l|l} \delta(c_1) = 8 & \delta(c_3) = 7 & \delta(c_5) = 6 & \delta(c_7) = 3 & \delta(c_9) = 12 & \delta(c_{11}) = 1 & \delta(c_{13}) = 7 \\ \delta(c_2) = 3 & \delta(c_4) = 9 & \delta(c_6) = 0 & \delta(c_8) = 3 & \delta(c_{10}) = 2 & \delta(c_{12}) = 4 & \delta(c_{14}) = 5 \end{array}$$

Le niveau d'implication de c est égal à $\max(s(k), s(\neg(k))) = 4$ qui est donné par le marquage du graphe illustré dans la figure ci-dessous.

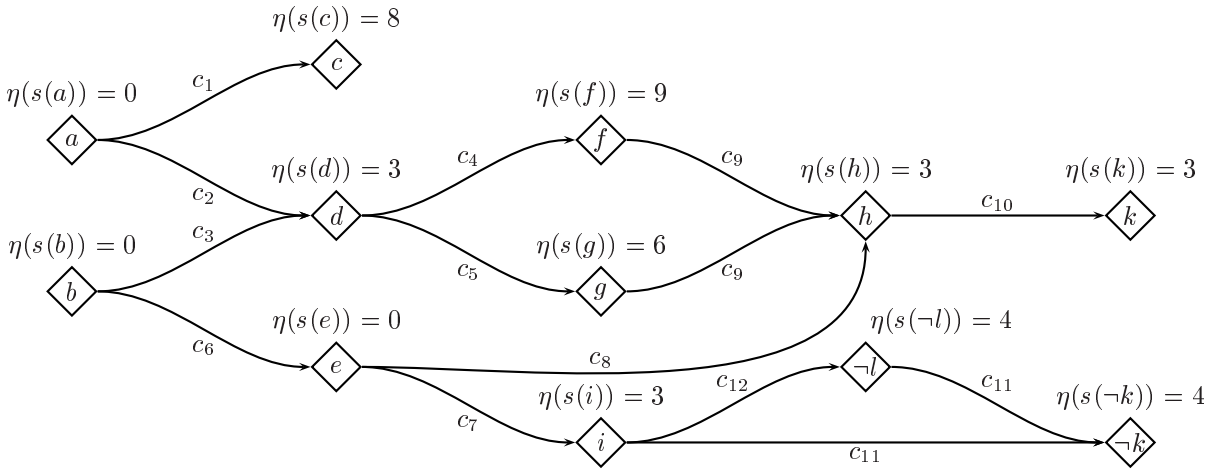

FIG. 7.16 – *Graphe et niveau d'implication d'une clause*

Nous proposons page suivante un algorithme permettant de calculer le niveau minimal d'implication d'une clause étant donné son graphe d'implication.

L'algorithme 7.8 nous permet de déterminer pour chaque clause testée, lors de l'application du théorème, son niveau minimal d'implication. Le niveau d'application du théorème de partition du modèle généralisé restreint à la propagation unitaire s'obtient, de manière évidente, en prenant le maximum des niveaux d'implication des différentes clauses impliquées. En conséquence, lorsque le théorème est prouvé s'appliquer jusqu'à un niveau n, toutes les branches situées entre le niveau n et le niveau actuel sont coupées.

Algorithme 7.8 NIVEAU MINIMAL D'IMPLICATION

```
1.  Function Niveau_Minimal_Implication : Integer
2.  Input  : Un graphe G = (S, A) et une clause c ;
3.  Output : Un entier correspondant au niveau minimal d'implication de c
             (l'infini si la clause n'est pas impliquée) ;
4.  Begin
5.    L = ∅ ;                                          %% Liste des sommets a examiner
6.    foreach sommet s ∈ S
7.    do
8.        η(s)=∞ ;
9.    done
10.   foreach littéral l t.q. ¬l ∈ c
11.   do
12.      η(s(l)) = 0 ;
13.      L = L ∪ {s | s = succ(s(l)) et s ∉ L} ;
14.   done
15.   while(L ≠ ∅)
16.   do
17.      s = le premier sommet de L ;
18.      L = L \ {s} ;
                                  %% La valeur d'un sommet est donnée par la valeur du
                                  %% plus petit chemin permettant de produire ce sommet
19.      foreach P_i ⊂ pred(s) t.q. ∀ p ∈ P_i, val(⟨p, s⟩) = c_i
20.      do            %% La valeur d'un chemin est donnée par le maximum entre le niveau
            %% d'apparition de la clause et la valeur de chaque sommet père pour ce chemin
21.         val_chemin = δ(c_i) ;
22.         foreach s' ∈ P_i
23.         do
24.             val_chemin = max(val_chemin, η(s')) ;
25.         done
26.         η(s) = min(η(s),val_chemin) ;
27.      done
28.      L = L ∪ {s'' | s'' = succ(s)} ;
29.   done
30.   C = {p | p est un couple de sommets (s(l), s(¬l)) t.q. succ(s(l)) = succ(s(¬l)) = ∅}
31.   niveau_implication=∞                  %% Le niveau d'implication de c est donné
                                %% par le minimum des chemins menant à une contradiction
32.   foreach (s(l), s(¬l)) ∈ C
33.   do                %% La valeur d'un chemin menant à une contradiction est donnée par
                                          %% le maximum valeurs de deux sommets opposés
34.      niveau_implication=min(niveau_implication, max(s(l), s(¬l))) ;
35.   done
36.   return niveau_implication ;
37. End
```

D'autre part, l'étiquetage des clauses à chaque niveau de l'arbre de Davis & Putnam nous permet de déterminer pour chaque clause son niveau d'apparition dans l'arbre de DP. L'implication d'une clause peut être due uniquement à des clauses apparues antérieurement au niveau actuel. Par exemple, si $c_1 = \{l, \alpha\}$ et $c_2 = \{\neg l, \alpha\}$ sont deux clauses de Σ, le niveau d'implication de α par $\Sigma_{l_{[1:i]}}$ est égal à zéro (c'est-à-dire la clause est impliquée par Σ).

Par ailleurs, nous avons vu précédemment (cf. alinéa 2° page 120) que chaque clause impliquée pouvait être ajoutée (en espace constant) à la base. Cet ajout se limitait jusqu'alors au traitement du nœud courant. L'extension précédente nous permet de conserver cette clause jusqu'à son niveau minimal d'implication, ce qui permet d'élaguer l'arbre de recherche.

Cette extension réalisée en fin de thèse est à l'heure actuelle en cours d'implantation. Cependant à la vue de l'ensemble des remarques faites ci-dessus, cette extension nous paraît extrêmement prometteuse, particulièrement pour résoudre des problèmes structurés.

De nombreuses techniques de simplification de DP utilisées actuellement peuvent être étendues en utilisant notre approche. Nous pensons en particulier à associer un niveau minimal d'implication à chaque littéral fixé (impliqué) par la propagation unitaire dans C-SAT, Satz, etc.

7.3 Conclusion

Notre contribution à la procédure de Davis & Putnam porte sur la mise au point d'une nouvelle technique d'élagage de l'arbre de recherche. D'un point de vue théorique, nous avons généralisé le théorème de partition du modèle proposé initialement dans [Jeannicot *et al.* 1988]. D'un point de vue pratique, nous avons proposé une nouvelle technique de simplification basée sur la restriction du théorème de partition du modèle généralisé à l'implication via la propagation unitaire.

Nous avons montré expérimentalement que cette technique permettait de réduire sensiblement le nombre de nœuds explorés par la procédure de Davis & Putnam. Cependant, notre procédure d'élagage ne permet pas pour l'instant de réduire le temps de calcul.

Les perspectives de ces travaux sont nombreuses. Dans un premier temps, il est nécessaire de diminuer le coût (en temps) de notre procédure d'élagage. Pour cela plusieurs pistes sont en cours d'exploration. Notamment, nous pensons que déterminer un ordre judicieux quant aux tests d'implications réalisés pour s'assurer de l'application du théorème, permettra de réduire significativement le temps de calcul. Une autre optimisation consiste à n'appliquer le théorème qu'à partir d'un certain niveau de l'arbre de DP, ceci afin de profiter d'un plus grand nombre de clauses binaires. D'autre part, la voie qui nous semble la plus prometteuse, est certainement l'introduction du concept de niveau d'implications. Une procédure de Davis & Putnam utilisant le théorème de partition du modèle généralisé restreint à la propagation unitaire et les niveaux d'implications est en cours d'implantation.

Par ailleurs, il nous semble que les cibles privilégiées d'un tel processus d'élagage sont les instances structurées. Ainsi, les gains déjà constatés sur les instances aléatoires devraient être accrus lors de tests portant sur des instances structurées. Les premiers résultats obtenus sur ce type d'instances confirment cette hypothèse.

Chapitre 8

DP et Recherche Locale : Combinaisons

L'efficacité des algorithmes de recherche locale est incontestablement supérieure à celle des algorithmes systématiques, lorsqu'il s'agit de vérifier la satisfaisabilité d'une instance consistante de **SAT**. Cependant, la recherche locale ne peut prouver a priori l'inconsistance. Par ailleurs, sur une instance de grande taille une recherche systématique, comme la procédure de Davis & Putnam, a peu de chance de trouver un modèle. Il est donc naturel de s'interroger sur la faisabilité d'une méthode permettant de faire coopérer des méthodes de recherche locale et des méthodes systématiques. Cette coopération a pour objectif de pallier les inconvénients de chacune des méthodes. Une telle combinaison apparaît, pour beaucoup de chercheurs, comme une voie extrêmement prometteuse pour la résolution pratique du problème **SAT** [Freuder *et al.* 1995, Selman *et al.* 1997].

À notre connaissance, peu de méthodes mixtes ont été développées à l'heure actuelle pour **SAT**. Nous proposons dans ce chapitre, de nouveaux schémas de combinaison entre méthodes de recherche locale et méthodes systématiques. Notamment, nous montrons comment une méthode de type recherche locale, logiquement restreinte à la recherche de solutions, peut être utilisée pour circonscrire l'inconsistance d'instances de **SAT**.

Nous commençons ce chapitre par une présentation des schémas de combinaison déjà existants dans la littérature. Dans la deuxième partie, nous proposons de distinguer deux types d'inconsistance que nous formalisons et opposons : les inconsistances locales et les inconsistances globales. Nous détaillons par la suite comment et dans quelle mesure une méthode de recherche locale peut permettre de localiser l'inconsistance. Puis, nous proposons plusieurs méthodes de coopération utilisant la recherche locale comme processus permettant de circonscrire l'inconsistance. Nous faisons notamment une description détaillée d'un algorithme mixte, que nous avons appelé DPRL (« Davis & Putnam - Recherche Locale »). Enfin avant de conclure, nous présentons les résultats expérimentaux obtenus par DPRL et décrivons quelques optimisations possibles.

8.1 Les schémas de combinaison existants

Il est naturel de chercher à combiner les qualités complémentaires des algorithmes complets et incomplets en se basant sur différents schémas de coopération. De nombreux travaux très récents concernent explicitement ou implicitement la recherche de coopération entre ces méthodes aux propriétés complémentaires dans le cadre des problèmes de satisfaction de contraintes (CSP). Un inventaire complet de ces

méthodes mixtes pour les CSP peut être trouvé dans [Lobjois & Lemaître 1997]. Cependant, de tels algorithmes pour **SAT** ne sont pas légion dans la littérature. Nous proposons un classement des schémas de coopération existants pour **SAT** selon la souche de la greffe, c'est-à-dire selon que l'on greffe une méthode incomplète sur l'ossature d'une méthode complète ou inversement.

8.1.1 Complet greffé sur incomplet

Cette catégorie regroupe les schémas de coopération dont la base est une méthode incomplète. Cette base est modifiée ou orientée par l'utilisation des mécanismes empruntés aux méthodes complètes.

8.1.1.1 Résolution alternée

L'archétype de ce schéma de combinaison est la « résolution alternée » [Zhang & Zhang 1996]. Cette méthode consiste en une assignation partielle et aléatoire des variables, par la suite une recherche systématique de type « backtrack » tente de résoudre le sous-problème induit par l'assignation partielle. Si la méthode systématique échoue, l'interprétation partielle est réparée en utilisant une stratégie de recherche locale et le processus est itéré. Dans [Kask & Dechter 1996] une autre forme de résolution alternée est présentée comme une amélioration de la méthode incomplète GSAT.

8.1.1.2 Ajouts de clauses

Un autre schéma de coopération consiste à rendre complète une méthode de recherche locale par ajouts de clauses. Plusieurs algorithmes utilisant ce procédé, ont été mis au point (voir par exemple le « *Dynamic Backtracking* » [Ginsberg 1993, Ginsberg & McAllester 1994] ou CH-SAT [Hao & Tétard 1996]).

8.1.1.3 SCORE(FD/B)

SCORE est un algorithme de CSP basé sur une approche « locale systématique » [Chabrier *et al.* 1991]. Nous nous intéressons ici à la version SCORE(FD/B) [Chabrier *et al.* 1995] spécialisée dans la résolution de **SAT**. Cet algorithme comporte deux phases : une phase d'initialisation et une phase de résolution.

L'objet de la phase d'initialisation est de construire une configuration initiale. Différentes procédures sont utilisables, l'une des plus efficaces consiste à assigner chaque variable en fonction de son déséquilibre [1], ceci afin de satisfaire le maximum de littéraux.

La phase de résolution consiste à effectuer une recherche locale systématique de solution à partir de la configuration initiale. L'arbre de recherche exploré pendant cette phase est celui d'un algorithme de type DP avec les spécificités suivantes :

– à chaque nœud de l'arbre de recherche est associé un ensemble de clauses de la même manière que DP, ainsi qu'une configuration courante obtenue en assignant chaque variable de cet ensemble de clauses avec la valeur assignée à cette variable dans la configuration initiale ;

1. Nous rappelons que le déséquilibre d'une variable est la différence entre ses occurrences positives et négatives.

– les heuristiques de choix des littéraux prennent en compte la configuration courante, c'est-à-dire l'ensemble de clauses associé au nœud simplifié par la configuration initiale, l'heuristique de base privilégie les littéraux apparaissant dans un nombre maximum de clauses contredites par la configuration courante et les littéraux minimisant le nombre de clauses contredites ;
– la fonction de détection de solutions retourne une réponse positive lorsque la configuration courante est une solution.

Cette approche permet, dans certains cas, une détection de solutions plus précoce qu'avec un Davis & Putnam « classique ».

8.1.2 Incomplet greffé sur complet

Nous rangeons dans cette catégorie les schémas de coopération dans lesquels la méthode complète est une souche sur laquelle on greffe des mécanismes incomplets. Dans ces méthodes, la plupart du temps, la souche complète est l'algorithme de Davis & Putnam (DP) et les mécanismes incomplets sont des algorithmes de recherche locale (RL).

8.1.2.1 Complétude partielle

Une méthode duale à la résolution alternée consiste à limiter la profondeur de l'arbre de DP et de continuer la recherche, au delà de cette profondeur, par une méthode de recherche locale. Le niveau à partir duquel la recherche locale est lancée peut varier selon le reliquat de temps disponible ou selon une estimation de la qualité de la meilleure solution espérée dans le sous-espace pendant.

Cette idée de complétude partielle selon la profondeur est utilisée dans [Génisson & Rauzy 1996]. R. Génisson et A. Rauzy montrent qu'un DP limité à une profondeur définie au départ de la procédure, reste complet sur des instances Horn-renommables ou sur des instances de la hiérarchie de G. Gallo et M.G. Scutellà (cf. paragraphe 3.3). Cependant, l'algorithme proposé dans [Génisson & Rauzy 1996] ne fait en aucune façon appel à un algorithme de recherche locale.

8.1.2.2 Pré-traiter pour ordonner

J. Crawford utilise une recherche locale où des poids sont affectés aux variables présentes dans les clauses qui semblent difficiles à satisfaire [Crawford 1993]. Ce poids permet ensuite d'ordonnancer les variables en vue d'un parcours systématique de l'espace de recherche. Les résultats expérimentaux obtenus par l'approche proposée par J. Crawford étaient extrêmement décevants comparativement aux résultats des approches plus classiques comme GSAT ou C-SAT. En effet, la validation expérimentale ayant été réduite aux instances aléatoires *3***SAT** au seuil, J. Crawford, sans voir l'intérêt réel d'une telle approche, a abandonné cette piste. Indépendamment de ces travaux, nous avons développé une approche similaire [Mazure *et al.* 1996a, Mazure *et al.* 1996b] et avons validé une telle approche sur un large panel d'instances de **SAT** issues de « *benchmarks* » traditionnels comme ceux proposés aux challenges de DIMACS [DIM1993] (cf. paragraphe 8.4.2).

8.1.3 Synthèse

La plupart des approches dites mixtes constituent une « amélioration » vis-à-vis des méthodes présentées dans les chapitres 6 et 7, dans le sens où elles tentent de faire coopérer les approches à base de réparations locales et les approches systématiques afin de profiter des avantages de chacune. Toutefois cette coopération reste bien souvent partielle. En effet par exemple, J. Crawford utilise la recherche locale pour définir un ordonnancement statique, mais une fois cet ordonnancement défini, la résolution via DP s'effectue classiquement sans qu'aucune autre interaction n'intervienne entre DP et la recherche locale ; ou encore la résolution alternée utilise successivement les deux approches mais jamais conjointement. La résolution mixte ne bénéficie donc pas réellement des qualités des deux approches en même temps mais plutôt successivement. Il n'y a donc pas réellement communication entre les approches : la coopération se limite à l'échange, lorsqu'il a lieu, d'au maximum une seule information durant tout le processus. Ces constats expliquent peut être pourquoi en pratique, ces méthodes n'égalent que très rarement les approches classiques.

Nous proposons de nouveaux schémas de coopération (cf. paragraphe 8.4) où l'accent est mis sur l'interaction entre les méthodes. Nous montrons la faisabilité pratique de telles méthodes et présentons de nombreux résultats expérimentaux où sur un large panel d'instances la coopération des méthodes complètes et incomplètes se révèle beaucoup plus efficace que les approches dites « classiques ».

8.2 Inconsistances locales vs. globales

La détection de l'inconsistance logique est un problème fondamental et crucial pour de nombreuses applications telles que la démonstration automatique, la révision de bases de connaissances, les problèmes de VLSI et de diagnostic, etc. Or, il est possible de distinguer plusieurs « formes » d'inconsistances :

Définition 8.1 (Inconsistance globale)
Une base de connaissances Σ *est* **globalement inconsistante**
si et seulement si
Σ *est inconsistante et* $\forall\ \Pi \subsetneq \Sigma$, Π *est consistante.*

Lorsqu'une base de connaissances n'est pas inconsistante globalement, elle est dite localement inconsistante.

Définition 8.2 (Inconsistance locale)
Une base de connaissances Σ *est* **localement inconsistante**
si et seulement si
Σ *est inconsistante et* $\exists\ \Pi \subsetneq \Sigma$, *t.q.* Π *est inconsistante.*

Par ailleurs, plusieurs degrés de localité peuvent être définis. Par exemple, il est possible de caractériser cette localité par le rapport entre la taille de la plus petite sous-base inconsistante et la taille de la base de connaissance initiale. Les sous-bases inconsistantes de la base sont appelées des noyaux inconsistants. Plus formellement, les notions de noyau inconsistant, noyau globalement inconsistant et noyau minimalement inconsistant se définissent comme suit.

Définition 8.3 (Noyau inconsistant, globalement inconsistant et minimalement inconsistant)
Soit Σ *une base de connaissance inconsistante.* Π *est appelé* **noyau inconsistant** *de* Σ *si et seulement si* $\Pi \subset \Sigma$ *et* Π *est inconsistant.*

On dit que Π *est* **noyau globalement inconsistant** *si et seulement si* Π *est globalement inconsistant.*
Un noyau inconsistant Π *de* Σ *est dit* **minimalement inconsistant** *si et seulement si* Π *est globalement inconsistant et pour tout noyau inconsistant* Ω *de* Σ, $|\Pi| \leqslant |\Omega|$.

Ces différentes formes d'inconsistance sont d'une importance capitale dans de nombreuses applications. En effet, très fréquemment l'inconsistance d'une base de connaissance est due à la présence accidentelle d'une information contradictoire sur un sujet donné. Cette inconsistance localisée pollue globalement la base de connaissance : toute information ainsi que son contraire peut être déduite de cette base. Malheureusement, il n'existe pas de méthode universelle et efficace pour détecter et/ou localiser ce genre d'inconsistance.

Par ailleurs, les problèmes inconsistants globalement sont certainement les plus difficiles à résoudre. En effet, pour ce type d'inconsistance toutes les propositions interviennent dans la contradiction ; la preuve de l'inconsistance est souvent difficile à obtenir en temps raisonnable. Il apparaît que la plupart des bases de connaissance globalement inconsistantes sont générées de manière artificielle et ne se retrouvent pas dans les applications réelles. En outre, très souvent, ce genre de problèmes possèdent de fortes régularités (*e.g.* le problème des « pigeons » [Cook 1971], les formules de G.S. Tseitin [Tseitin 1968], etc.) qui permettent leur résolution par des méthodes spécialisées.

8.3 Détection de noyaux inconsistants

Nous explicitons dans ce paragraphe comment une méthode de type recherche locale peut être utilisée pour localiser l'inconsistance d'instances de **SAT**. C'est-à-dire dans quelle mesure est-il possible d'exploiter le travail effectué par la recherche locale pour la preuve de l'inconsistance ?

Afin de répondre à cette question, nous commençons notre étude par une série de tests des méthodes de recherche locale sur des problèmes inconsistants. Pour ces expérimentations nous avons utilisé TSAT (cf. paragraphe 6.2). Les problèmes étant inconsistants, TSAT ne pouvait qu'échouer dans sa tentative de trouver un modèle. À la fin de la recherche, nous avons totalisé pour chaque littéral le nombre d'apparitions dans une clause falsifiée et pour chaque clause le nombre de fois où elle a été falsifiée.

Intuitivement, les clauses les plus souvent falsifiées ont une forte chance de constituer un noyau inconsistant ; de même, les littéraux étant apparus le plus souvent dans les clauses falsifiées ont une forte chance d'appartenir à l'ensemble des littéraux d'un noyau inconsistant.

Cette hypothèse s'est très souvent révélée expérimentalement correcte. Plus précisément, lorsque l'on essaie de résoudre un problème localement inconsistant au moyen d'une méthode à base de réparations locales, la trace du nombre de fois où chaque clause a été falsifiée permet de faire émerger un noyau inconsistant. Par ailleurs, plus faible est le rapport entre la taille du noyau et la taille de l'instance, plus fort est l'écart entre les scores des clauses appartenant au noyau et les scores des clauses n'intervenant pas dans l'inconsistance.

Pour illustrer notre propos, nous avons fusionné les représentations clausales d'un problème inconsistant à savoir le problème des « 8 pigeons » (cf. exemple 5.1) (56 variables et 204 clauses), et d'un problème consistant bien connu, le problème des « 8 reines[2] » (64 variables et 736 clauses). La représentation de chacun des problèmes contient deux types de clauses : des clauses positives (uniquement constituées de littéraux positifs) et des clauses binaires négatives (constituées de deux littéraux négatifs).

2. Ce problème consiste à placer N reines sur un échiquier $N \times N$, sans qu'aucune des reines ne se prenne mutuellement.

La figure suivante (cf. FIG. 8.1) montre pour chaque clause le nombre de fois où elle a été falsifiée ($\#falsifiées$). Il apparaît une séparation aiguë et nette qui permet de retrouver clairement les deux problèmes (pigeons et reines) dans l'instance fusionnée.

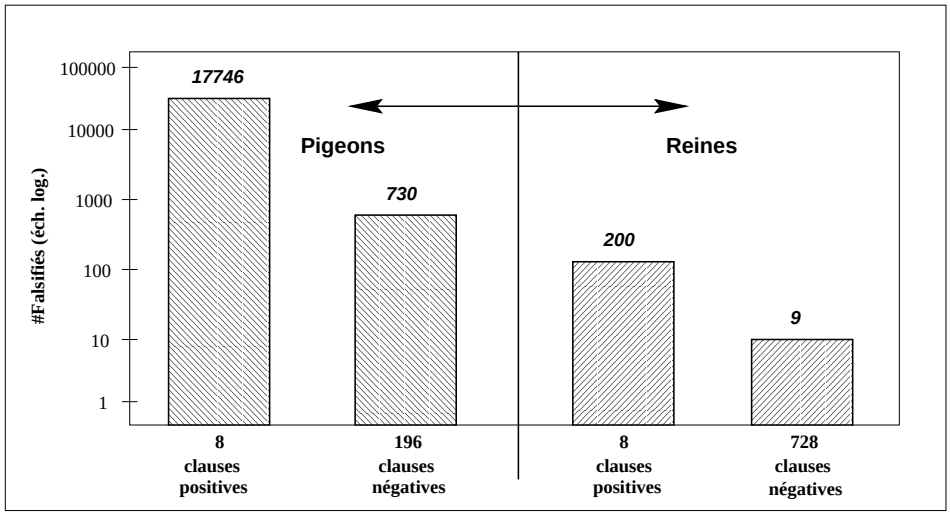

FIG. 8.1 – *Trace de TSAT sur le problème fusionné* « *8 pigeons* » *et* « *8 reines* »

Dans cet exemple, nous constatons que chaque type de clauses (*e.g.* les clauses positives du problème des « pigeons ») présente des scores très proches. Dans le diagramme de la présente page, nous avons mentionné le score moyen de chaque type de clauses. Ces scores sont compris dans les intervalles $[17189...18003]$, $[639...828]$, $[128...230]$ et $[1..38]$. Ils ont été obtenus en utilisant TSAT avec les valeurs de paramètres suivants : `Max_Tries`$= 20$ et `Max_Flips`$= V^2(V = 120\ (56 + 64))$. De plus, si l'on augmente les ressources allouées à TSAT, on constate que non seulement l'écart entre chaque type de clauses s'accentue mais également que les intervalles se rétrécissent.

Les résultats obtenus montrent que TSAT a permis de mettre en évidence les propriétés structurelles et syntaxiques de chacun des problèmes « pigeons » et « reines ». Pour des problèmes issus d'applications réelles, une séparation nette entre le score des clauses a permis également de mettre en évidence un noyau inconsistant du problème. La trace suivante (cf. FIG. 8.2 page ci-contre) montre les scores obtenus par TSAT sur un problème de vérification de circuit proposé au challenge DIMACS par A.V. Gelder [DIM1993]. Ce problème (`bf1355-638`) est constitué de 6768 clauses construites sur 2177 variables propositionnelles et a été prouvé inconsistant.

La trace de TSAT a permis sur ce problème d'obtenir un noyau minimalement inconsistant de 154 clauses construites à l'aide 83 variables propositionnelles ! Il est clair, au vu de ces résultats, qu'une méthode ne tenant pas du tout compte de cette notion de localité d'inconsistance pourra être conduite à résoudre la partie inconsistante du problème autant de fois qu'il existe de solutions pour la partie consistante. Cette trace de TSAT nous semble donc importante et nous montrons dans le prochain paragraphe comment l'utiliser efficacement dans le cadre d'un algorithme mixte.

8.4 De nouveaux schémas de combinaison

Des résultats précédents, nous pouvons déduire que l'utilisation de méthodes de recherche locale permet de circonscrire l'inconsistance d'instances de **SAT**. Nous proposons trois nouvelles manières de combiner

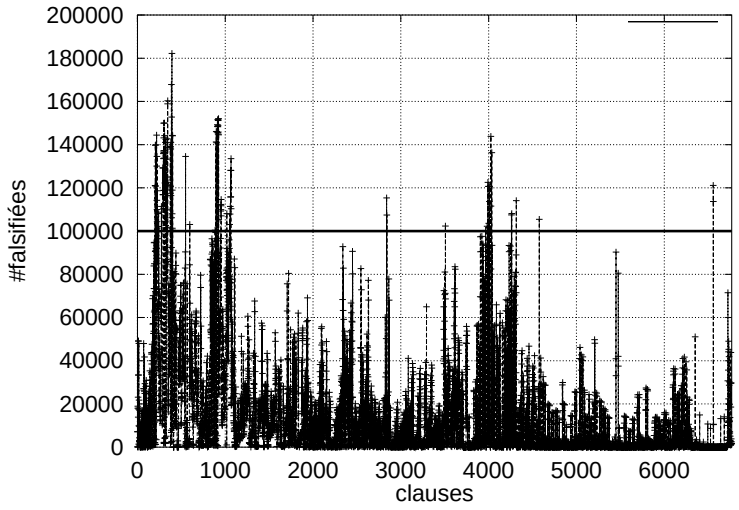

FIG. 8.2 – *Trace de TSAT sur le problème de circuit « bf1355-638 »*

un algorithme de recherche locale (RL) avec la procédure de Davis & Putnam (DP) :

1. incrémentale (utilisation de la RL et de DP en alternance) ;
2. pré-traitement (utilisation de la RL à la racine de l'arbre de DP) ;
3. heuristique (utilisation de la RL comme heuristique de branchement pour DP).

Le premier schéma de combinaison proposé appartient à la catégorie des méthodes à souche incomplète dans la classification faite précédemment. À l'opposé les approches pré-traitement et heuristique utilisent un DP comme ossature.

8.4.1 Approche incrémentale

L'objectif de cette méthode (DPRL-INCR) est de construire incrémentalement un noyau inconsistant. L'idée est d'utiliser la recherche locale comme un outil permettant de scinder l'instance initiale en deux parties, dont l'une au moins est inconsistante.

DPRL-INCR commence par effectuer une recherche locale. Dans le cas où cette recherche aboutit à un modèle, l'algorithme s'arrête en ayant prouvé la satisfaisabilité de l'instance. Dans le cas contraire, l'instance a plus de chance d'être contradictoire, nous utilisons les scores des clauses (c'est-à-dire le nombre de fois qu'elles ont été falsifiées durant la recherche locale) pour scinder l'instance Σ en deux parties : une première constituée des clauses faiblement falsifiées et une seconde constituée des clauses fortement falsifiées Σ_{fort}. Ensuite la satisfaisabilité de Σ_{fort} est testée par la procédure de Davis & Putnam classique. Si DP prouve l'inconsistance de Σ_{fort}, l'algorithme s'arrête possèdant ainsi la preuve que Σ est inconsistant. Si DP exhibe un modèle de Σ_{fort}, l'algorithme DPRL-INCR est relancé en augmentant le nombre de clauses constituant Σ_{fort}. L'algorithme 8.1 détaille précisément DPRL-INCR.

DPRL-INCR prouve l'inconsistance d'une formule en exhibant un noyau inconsistant (Σ_{fort}) de la base de connaissance. À partir de ce noyau inconsistant, il est possible d'extraire un noyau globalement inconsistant. Cette opération coûteuse en temps est décrite dans l'algorithme 8.2. Elle consiste à supprimer

les clauses (dans un certain ordre fixé) ne participant pas à l'inconsistance.

Algorithme 8.1 DPRL-INCR

```
1.  Function DPRL-incr : boolean
2.  Input  : Σ un ensemble de clauses, N le nombre de clauses de Σ_fort ;
3.  Output : true si Σ admet un modèle, false sinon  ;
4.  Begin
5.     if (RL(Σ)) then return true ;
6.     else
7.        Σ_trie=les clauses de Σ triées suivant leur ordre décroissant de score ;
8.        Σ_fort=∅ ;
9.        for i from 1 to N do
10.          Σ_fort=Σ_fort ∪ (la i ème clause de Σ_trie) ;
11.       done
12.       if (Σ_fort==Σ) then return DP(Σ) ;
13.       elif (DP(Σ_fort))  then return DPRL-incr(Σ,N+pas) ;
14.       else return false ;
15.       fi
16.    fi
17. End
```

Seuls ces multiples tests de consistance permettent de garantir l'obtention d'un noyau globalement inconsistant. Il est clair que l'ordre de parcours des clauses est primordial pour « l'efficacité » de cet algorithme. Nous pensons que l'ordre induit par les méthodes de recherche locale est le meilleur à l'heure actuelle. En effet, c'est le seul à notre connaissance qui fait émerger des informations sémantiques permettant de circonscrire les clauses critiques d'une base de connaissance.

La conjugaison de l'algorithme 8.1 qui fournit la plupart du temps un noyau inconsistant de taille très réduite par rapport à la taille de l'instance initiale, et de l'algorithme 8.2 qui grâce aux méthodes de recherche locale, fournit un ordre judicieux de parcours des clauses, a permis d'obtenir de manière relativement efficace des noyaux globalement inconsistants pour la majorité des instances inconsistantes de DIMACS (cf. paragraphe 8.5).

Algorithme 8.2 RENDRE GLOBALEMENT INCONSISTANT

```
1.  Function Rendre_Globalement_Inconsistant : un ensemble de clauses
2.  Input  : Σ un noyau inconsistant ;
3.  Output : un noyau globalement inconsistant ;
4.  Begin
5.     RL(Σ) ;                    %% Lors de RL, les scores des clauses sont totalisées
6.     Ω=Σ ;
7.     foreach clause c de Σ par ordre croissant des scores do
8.        Ω=(Ω / {c}) ;                              %% Supprimez la clause c de Ω
9.        if (DP(Ω)) then Ω=(Ω ∪ {c}) ;          %% c participe à la contradiction
10.    done
11.    return Ω ;
12. End
```

8.4.2 Approche pré-traitement

L'exploitation que nous faisions jusqu'à maintenant de la trace des algorithmes de recherche locale est basée sur le score des clauses. Nous discutons dans ce paragraphe d'un schéma de combinaison exploitant les scores obtenus par les littéraux. Par ailleurs, contrairement à l'algorithme DPRL-INCR, la combinaison détaillée dans cette section repose sur DP.

Nous avons vu dans le chapitre précédent que l'efficacité de DP repose sur un choix judicieux des littéraux à affecter à chaque nœud de l'arbre de recherche. Par ailleurs lors d'un échec de la RL, nous pensons que les littéraux étant apparus le plus souvent dans les clauses falsifiées ont une forte chance d'appartenir à un noyau inconsistant (cf. paragraphe 8.3). Un schéma de combinaison naturelle consiste à ce que l'heuristique de branchement de DP choisisse prioritairement les littéraux apparaissant le plus fréquemment dans des clauses falsifiées lors de la RL. Un tel schéma devrait permettre à DP de prouver très rapidement qu'un sous-ensemble des clauses est contradictoire.

La recherche locale est utilisée dans ce schéma comme un pré-traitement à la procédure de Davis & Putnam. Ce pré-traitement permet d'obtenir un ordre total sur les variables, cet ordre étant induit par leur nombre d'apparitions dans les clauses falsifiées. L'ordre est ensuite exploité au sein de DP : lors d'un point de choix, c'est la variable (non encore affectée) possédant le meilleur score suivant l'ordre qui est choisie. Cette méthode que nous avons baptisée DPRL-PRÉ a été proposée indépendamment par J. Crawford dans une version très similaire au second challenge DIMACS [Crawford 1993]. J. Crawford a dédié son algorithme à la résolution d'instances *3***SAT** au seuil. Cependant les instances aléatoires au seuil n'étant que très peu localement inconsistantes, voire même globalement inconsistantes pour un nombre de variables très grand (loi $0-1$), aucun résultat probant n'a pu être obtenu. De ce constat d'échec, J. Crawford a abandonné cette piste, sans avoir constaté qu'une telle méthode pouvait permettre de localiser des noyaux inconsistants d'instances de **SAT** autres qu'aléatoires. L'algorithme 8.3 décrit la procédure DPRL-PRÉ.

Algorithme 8.3 DPRL-PRÉ

```
1.  Function DP-ORDRE : boolean
2.  Input  : Σ un ensemble de clauses, Ordre la liste triée des variables Σ ;
3.  Output : true si Σ admet un modèle, false sinon  ;
4.  Begin
5.     Σ=Propagation_Unitaire(Σ) ;                          %% cf. algorithme 7.2
6.     Σ=Simplification_Littéraux_Purs(Σ) ;                 %% cf. algorithme 7.3
7.     if (Σ contient la clause vide) then return false ;
8.     elif (Σ==∅) then return true ;
9.     else v=la première variable de Ordre apparaissant dans Σ ;
10.         return ((DP-ordre(Σ ∧ {v},Ordre)) ∨ (DP-ordre(Σ ∧ {¬v},Ordre))) ;
11.    fi
12. End

13. Function DPRL-PRÉ : boolean
14. Input  : Σ un ensemble de clauses ;
15. Output : true si Σ admet un modèle, false sinon  ;
16. Begin
17.    if (RL(Σ) then return true ;
18.    else foreach variable v do Calculer score[v] ; done
 %% score[v] correspond au nombre d'apparitions de v dans des clauses falsifiées durant RL
19.         Ordre=liste des var. de Σ triées par ordre décroissant de leur score ;
20.         return DP-ordre(Σ,Ordre) ;
21.    fi
22. End
```

Sur des instances **SAT** où l'inconsistance est extrêmement localisée (c'est-à-dire où le noyau inconsistant est de très petite taille) comme par exemple les instances AIM (voir l'annexe 13 pour une description de ces instances), cette approche est d'une efficacité incontestablement supérieure à un DP classique. Cependant, ce schéma de combinaison ne tient en aucune façon compte de l'évolution de la base de connaissance au cours de la recherche de DP. En effet, les informations obtenues par la recherche locale sur l'éventuel noyau inconsistant sont obtenues à la racine de l'arbre de DP, c'est-à-dire sur la base Σ et non pas sur $\Sigma_{\{l_1,\ldots,l_n\}}$. Il est donc légitime de s'interroger sur la pertinence de l'ordre obtenu à la racine de l'arbre, tout au long de DP. Il nous semble que les « contre-performances » obtenues par cette méthode mixte sur certaines instances inconsistantes sont dues précisément à cette rigidité de l'ordre.

8.4.3 Approche heuristique

L'algorithme DPRL [Mazure *et al.* 1998a] est un algorithme complet se basant sur la procédure DP et utilisant la RL à la fois comme heuristique (choix de la prochaine variable à affecter) et comme un moyen de prolonger l'interprétation partielle effectuée par DP vers un modèle. L'idée de DPRL est proche de celle DPRL-PRÉ : les scores des variables sont toujours utilisés pour déterminer la prochaine variable à propager dans DP, mais ces scores sont renouvelés à chaque point de choix. Plus précisément, à chaque nœud de l'arbre de recherche construit par DP, si l'ensemble de clauses associé à ce nœud ne possède ni littéraux unitaires ni littéraux purs, nous appliquons une RL sur le problème simplifié par la branche courante de DP. Au retour de l'appel à la RL, deux cas se présentent :

1. la RL a exhibé un modèle du sous-problème induit par DP, dans ce cas l'algorithme DPRL s'arrête, la formule est alors satisfaisable (l'interprétation partielle décrite par la branche de DP est prolongée par le modèle exhibé par RL, pour obtenir un modèle de la base de connaissance initiale) ;
2. la RL a échoué dans la résolution du problème simplifié, dans ce cas le littéral étant apparu le plus souvent dans les clauses falsifiées lors de la RL est élu par DP comme prochaine variable à assigner.

L'algorithme DPRL est détaillé ci-dessous.

Algorithme 8.4 DPRL

```
1.  Function DPRL : boolean
2.  Input  : Σ un ensemble de clauses ;
3.  Output : true si Σ admet un modèle, false sinon  ;
4.  Begin
5.     Σ=Propagation_Unitaire(Σ) ;                          %% cf. algorithme 7.2
6.     Σ=Simplification_Littéraux_Purs(Σ) ;                 %% cf. algorithme 7.3
7.     if (Σ contient la clause vide) then return false ;
8.     elif (Σ==∅) then return true ;
9.     elif (RL(Σ)) then return true ;
10.    else
11.      l = le littéral apparaissant le plus souvent dans des clauses falsifiées
             lors de la recherche d'un modèle par RL ;
12.      return ((DPRL(Σ ∧ {l})) ∨ (DPRL(Σ ∧ {¬l}))) ;
13.    fi
14. End
```

Les nombreux appels à RL entrainent un surcoût non négligeable de temps par rapport à DP-ordre (cf l'algorithme 8.3 concernant DPRL-PRÉ), cependant c'est le prix à payer pour maintenir dynamiquement les scores des littéraux. Par ailleurs, dans DPRL-PRÉ afin d'obtenir le « meilleur » ordre il est nécessaire d'allouer un maximum de ressources à la procédure de RL, autrement dit la justesse de l'ordre dépend fortement du temps passé dans la recherche locale. Dans DPRL, seule la variable ayant obtenu le score le plus élevé lors de l'appel à RL est retenue. Une information correcte de ce type peut être obtenue lors d'un appel peu coûteux à RL. Cependant, il reste que par rapport à une heuristique classique, de type purement syntaxique (*e.g.* Jeroslow-Wang), l'heuristique RL pour DP est plus consommatrice en temps. Pour que l'algorithme soit en pratique exploitable, il faut que DPRL compense le « temps perdu » dans la recherche locale, par un arbre sensiblement plus petit que celui d'un DP classique. Cependant, la RL n'est pas utilisée comme une simple heuristique pour DP, elle conserve sa fonction première qui est d'exhiber un modèle. Ainsi sur certaines instances satisfaisables, DPRL peut être au moins aussi efficace qu'un algorithme classique de recherche locale et a fortiori plus efficace qu'un DP classique.

À notre connaissance, DPRL est le premier schéma de coopération permettant d'exploiter conjointement les qualités des approches complètes et incomplètes. Nous montrons dans le paragraphe suivant qu'un tel algorithme est tout à fait réalisable en pratique et que de plus, il est étonnamment efficace.

8.5 DPRL : résultats expérimentaux

Notre objectif principal au travers de ces expérimentations est de tester la faisabilité d'une méthode mixte telle que DPRL, c'est-à-dire de vérifier l'effet des variables fournies par la recherche locale sur la taille de l'arbre de recherche de DP.

Afin de ne pas biaiser les résultats, nous avons comparé notre approche avec un DP « classique », ne mettant pas en œuvre diverses stratégies de simplification implantées dans les meilleurs DP actuels (*e.g.* la résolution bornée mise en place dans C-SAT). En effet ces diverses stratégies que l'on rencontre dans toutes les meilleures versions de DP (C-SAT, POSIT, Tableau, Satz, ...) peuvent aisément être greffées à la procédure DPRL.

Pour notre comparaison, nous avons utilisé un Davis & Putnam employant l'heuristique « *First Fail In Shortened (ffis)* » proposée dans [Rauzy 1994]. Cette heuristique est une amélioration de l'heuristique traditionnelle Jeroslow-Wang (cf. paragraphe 7.1.3.2). Elle consiste à choisir prioritairement les variables ayant le maximum d'occurrences dans les clauses raccourcies et les plus courtes. En ce qui concerne DPRL, l'algorithme de recherche locale utilisé est TSAT (cf. paragraphe 6.2). DP+ffis et DP+TSAT ont été implantées sur une plate-forme commune [Mazure *et al.* 1996d, Mazure *et al.* 1998b].

Nos tests ont porté sur la quasi totalité des instances proposées au challenge DIMACS [DIM1993] (cf. annexe 13). Volontairement, les instances aléatoires *k***SAT** au seuil ont été exclues de nos expérimentations. En effet, le phénomène de seuil constaté sur ces instances tend à faire penser que les instances insatisfaisables générées au pic de difficulté sont globalement inconsistantes : il suffit de supprimer une clause au hasard pour que cette instance devienne satisfaisable. Or comme nous l'avons détaillé dans les sections précédentes, la cible privilégiée de nos approches est pour l'instant restreinte aux instances localement inconsistantes, du fait que la recherche locale permet de manière naturelle de faire ressortir ce noyau inconsistant. Nous avons montré expérimentalement que la plupart des instances inconsistantes de DIMACS satisfont ce critère de localité. Nous pensons que, dans les problèmes réels lorsqu'une inconsistance est présente dans la base de connaissances, cette inconsistance est souvent due à un petit sous-ensemble de faits en conflit polluant ainsi toute la base de connaissances. Cette hypothèse est avérée sur la quasi totalité des instances de DIMACS issues de problèmes réels (comme par exemple la vérification de circuits) ; ces instances comportent toutes un noyau inconsistant de taille relativement petite par rapport à la taille de

l'instance initiale.

La table 8.1 détaille pour chacune des instances testées, sa satisfaisabilité, son nombre de variables, son nombre de clauses et le nombre de nœuds nécessaires à DP+ffis et à DP+TSAT pour résoudre cette instance. Les temps de calculs ont également été précisés ; ces temps correspondent à une exécution sur un PC équipé d'un processeur Pentium 133 Mhz fonctionnant sous Linux. D'autre part, pour chacune des instances inconsistantes testées, nous avons calculé par l'intermédiaire de la procédure `Rendre_Globalement_Inconsistant` (cf. algorithme 8.2) la taille (le nombre de variables et le nombre de clauses) d'un des noyaux globalement inconsistants de l'instance. Cette taille figure également dans la table 8.1.

Dans la table suivante :

- « Noy. Inc. » signifie noyau inconsistant, cette colonne contient la taille d'un noyau globalement inconsistant ;
- « $> n$h » siginifie que l'algorithme n'a pu résoudre l'instance en moins de n heures de temps CPU ;
- « *** » dans :
  - la colonne « Noy. Inc. » siginifie que l'instance est satisfaisable, si l'instance est insatisfaisable cette colonne contient la taille d'un noyau globalement inconsistant ;
  - dans les autres colonnes signifie que l'algorithme a échoué dans sa tentative de résolution ;
- « ??? » signifie que la procédure `Rendre_Globalement_Inconsistant` n'a pas réussi au bout du temps imparti (1 semaine) à isoler un noyau globalement inconsistant.

Instances de	Taille		Noy. Inc.		DP+ffis		DP+TSAT	
DIMACS	V	C	V	C	#Nœuds	temps	#Nœuds	temps
aim-100-1_6-no-1	100	160	43	47	323296	50s30	13	0s22
aim-100-1_6-no-2	100	160	46	52	167445	23s38	19	0s34
aim-100-1_6-no-3	100	160	51	57	2E+06	214s71	16	0s26
aim-100-1_6-no-4	100	160	43	48	728908	97s59	14	0s25
aim-100-1_6-yes1-1	100	160	***	***	6663	1s09	8	0s12
aim-100-1_6-yes1-2	100	160	***	***	30052	4s39	6	0s08
aim-100-1_6-yes1-3	100	160	***	***	34	0s01	12	0s18
aim-100-1_6-yes1-4	100	160	***	***	7707	1s46	6	0s08
aim-100-2_0-no-1	100	200	18	19	2E+06	349s52	5	0s10
aim-100-2_0-no-2	100	200	35	39	1E+06	294s09	9	0s20
aim-100-2_0-no-3	100	200	25	27	394649	80s77	6	0s12
aim-100-2_0-no-4	100	200	26	31	1E+06	233s29	9	0s19
aim-100-2_0-yes1-1	100	200	***	***	31274	7s41	8	0s15
aim-100-2_0-yes1-2	100	200	***	***	10305	2s79	10	0s20
aim-100-2_0-yes1-3	100	200	***	***	10	0s00	6	0s12
aim-100-2_0-yes1-4	100	200	***	***	18	0s00	9	0s13
aim-100-3_4-yes1-1	100	340	***	***	30	0s02	2	0s08
aim-100-3_4-yes1-2	100	340	***	***	74	0s06	1	0s00
aim-100-3_4-yes1-3	100	340	***	***	193	0s14	1	0s01
aim-100-3_4-yes1-4	100	340	***	***	17	0s01	1	0s02
aim-100-6_0-yes1-1	100	600	***	***	4	0s01	1	0s01
aim-100-6_0-yes1-2	100	600	***	***	7	0s01	1	0s00
aim-100-6_0-yes1-3	100	600	***	***	11	0s01	1	0s01
aim-100-6_0-yes1-4	100	600	***	***	11	0s01	1	0s00
aim-200-1_6-no-1	200	320	52	55	***	>8h	16	0s58
aim-200-1_6-no-2	200	320	77	80	***	>15h	24	0s88
aim-200-1_6-no-3	200	320	77	83	***	>8h	33	1s15
aim-200-1_6-no-4	200	320	44	46	***	>17h	16	0s61
aim-200-1_6-yes1-1	200	320	***	***	10	0s01	10	0s25

Instances de **DIMACS**	Taille V	Taille C	Noy. Inc. V	Noy. Inc. C	DP+ffis #Nœuds	DP+ffis temps	DP+TSAT #Nœuds	DP+TSAT temps
aim-200-1_6-yes1-2	200	320	***	***	34	0s02	14	0s44
aim-200-1_6-yes1-3	200	320	***	***	***	>9h	11	0s32
aim-200-1_6-yes1-4	200	320	***	***	3E+07	5567s85	13	0s42
aim-200-2_0-no-1	200	400	49	53	***	>15h	11	0s47
aim-200-2_0-no-2	200	400	46	50	***	>8h	15	0s67
aim-200-2_0-no-3	200	400	35	37	***	>15h	10	0s42
aim-200-2_0-no-4	200	400	36	42	***	>8h	13	0s55
aim-200-2_0-yes1-1	200	400	***	***	7E+07	21859s45	27	1s21
aim-200-2_0-yes1-2	200	400	***	***	7E+06	2809s87	29	1s18
aim-200-2_0-yes1-3	200	400	***	***	217	0s09	20	0s98
aim-200-2_0-yes1-4	200	400	***	***	11783	4s79	27	1s07
aim-200-3_4-yes1-1	200	680	***	***	5409	6s88	1	0s06
aim-200-3_4-yes1-2	200	680	***	***	272	0s37	1	0s07
aim-200-3_4-yes1-3	200	680	***	***	36	0s05	3	0s22
aim-200-3_4-yes1-4	200	680	***	***	6270	8s44	1	0s05
aim-200-6_0-yes1-1	200	1200	***	***	75	0s25	1	0s01
aim-200-6_0-yes1-2	200	1200	***	***	134	0s47	1	0s01
aim-200-6_0-yes1-3	200	1200	***	***	214	0s72	1	0s01
aim-200-6_0-yes1-4	200	1200	***	***	232	0s72	1	0s04
aim-50-1_6-no-1	50	80	20	22	895	0s09	8	0s06
aim-50-1_6-no-2	50	80	28	32	782	0s10	9	0s07
aim-50-1_6-no-3	50	80	28	31	4525	0s41	12	0s09
aim-50-1_6-no-4	50	80	18	20	447	0s05	7	0s06
aim-50-1_6-yes1-1	50	80	***	***	84	0s01	6	0s05
aim-50-1_6-yes1-2	50	80	***	***	384	0s05	3	0s02
aim-50-1_6-yes1-3	50	80	***	***	5	0s01	4	0s04
aim-50-1_6-yes1-4	50	80	***	***	5	0s01	2	0s01
aim-50-2_0-no-1	50	100	21	22	2759	0s43	5	0s05
aim-50-2_0-no-2	50	100	28	30	974	0s17	7	0s07
aim-50-2_0-no-3	50	100	22	28	814	0s13	6	0s06
aim-50-2_0-no-4	50	100	18	21	1645	0s24	6	0s06
aim-50-2_0-yes1-1	50	100	***	***	176	0s03	3	0s03
aim-50-2_0-yes1-2	50	100	***	***	29	0s01	1	0s00
aim-50-2_0-yes1-3	50	100	***	***	446	0s07	3	0s02
aim-50-2_0-yes1-4	50	100	***	***	8	0s00	1	0s00
aim-50-3_4-yes1-1	50	170	***	***	20	0s01	1	0s00
aim-50-3_4-yes1-2	50	170	***	***	13	0s01	1	0s00
aim-50-3_4-yes1-3	50	170	***	***	16	0s01	1	0s01
aim-50-3_4-yes1-4	50	170	***	***	13	0s00	1	0s00
aim-50-6_0-yes1-1	50	300	***	***	4	0s01	1	0s00
aim-50-6_0-yes1-2	50	300	***	***	8	0s01	1	0s01
aim-50-6_0-yes1-3	50	300	***	***	4	0s01	1	0s00
aim-50-6_0-yes1-4	50	300	***	***	4	0s01	1	0s00
bf0432-007	1040	3668	674	1252	6E+06	19553s44	870	85s25
bf1355-075	2180	6778	82	185	2047	18s88	28	26s23
bf1355-638	2177	4768	83	154	***	>17h	32	32s57
bf2670-001	1393	3434	79	139	***	>25h	4822	519s40
ssa0432-003	435	1027	306	320	1570	1s79	16	0s80
ssa2670-130	1359	3321	552	669	***	>33h	79426	8040s64
ssa2670-141	986	2315	579	1247	2E+06	6350s77	92421	6639s44
ssa6288-047	10410	34238	???	???	***	>24h	***	>24h

Instances de	Taille		Noy. Inc.		DP+ffis		DP+TSAT	
DIMACS	V	C	V	C	#Nœuds	temps	#Nœuds	temps
ssa7552-038	1501	3575	***	***	***	>13h	1	0s34
ssa7552-158	1363	3034	***	***	78	0s19	1	0s29
ssa7552-159	1363	3032	***	***	84	0s21	1	0s25
ssa7552-160	1391	3126	***	***	76	0s18	1	0s30
jnh1	100	850	***	***	53	0s16	1	0s01
jnh10	100	850	67	85	44	0s19	9	0s85
jnh11	100	850	87	137	314	1s26	16	1s22
jnh12	100	850	***	***	49	0s23	1	0s02
jnh13	100	850	96	168	72	0s33	13	1s33
jnh14	100	850	90	148	26	0s12	10	0s85
jnh15	100	850	94	156	40	0s19	30	2s58
jnh16	100	850	99	282	599	2s36	330	25s90
jnh17	100	850	***	***	17	0s04	1	0s01
jnh18	100	850	95	200	137	0s59	64	5s36
jnh19	100	850	86	125	52	0s27	30	2s50
jnh2	100	850	74	86	35	0s17	2	0s20
jnh20	100	850	81	105	56	0s24	9	0s88
jnh201	100	800	***	***	27	0s05	1	0s01
jnh202	100	800	68	79	62	0s28	3	0s26
jnh203	100	800	92	155	70	0s32	33	2s97
jnh204	100	800	***	***	370	1s27	1	0s04
jnh205	100	800	***	***	84	0s27	1	0s01
jnh206	100	800	98	194	249	0s88	32	2s21
jnh207	100	800	***	***	25	0s08	1	0s04
jnh208	100	800	90	156	59	0s26	28	2s17
jnh209	100	800	***	***	64	0s22	1	0s02
jnh210	100	800	***	***	25	0s09	1	0s01
jnh211	100	800	64	84	37	0s16	10	0s85
jnh212	100	800	***	***	278	0s99	1	0s05
jnh213	100	800	***	***	41	0s11	1	0s02
jnh214	100	800	87	140	87	0s32	26	1s80
jnh215	100	800	95	154	247	1s03	11	0s79
jnh216	100	800	95	156	74	0s32	34	2s95
jnh217	100	800	***	***	24	0s06	1	0s01
jnh218	100	800	***	***	13	0s04	1	0s01
jnh219	100	800	94	177	145	0s75	52	3s91
jnh220	100	800	***	***	36	0s10	1	0s03
jnh3	100	850	96	189	107	0s44	81	5s20
jnh301	100	900	***	***	73	0s31	1	0s07
jnh302	100	900	21	22	22	0s13	1	0s11
jnh303	100	900	90	145	39	0s25	27	2s74
jnh304	100	900	79	106	42	0s21	4	0s41
jnh305	100	900	85	137	28	0s15	14	1s54
jnh306	100	900	99	202	384	1s67	72	6s28
jnh307	100	900	38	40	26	0s13	2	0s21
jnh308	100	900	94	169	78	0s37	16	1s68
jnh309	100	900	85	125	8	0s05	5	0s50
jnh310	100	900	12	13	32	0s17	2	0s23
jnh4	100	850	79	117	21	0s10	15	1s32
jnh5	100	850	67	83	36	0s15	7	0s76
jnh6	100	850	92	155	52	0s25	17	1s57
jnh7	100	850	***	***	15	0s04	1	0s02

Instances de **DIMACS**	Taille V	Taille C	Noy. Inc. V	Noy. Inc. C	DP+ffis #Nœuds	DP+ffis temps	DP+TSAT #Nœuds	DP+TSAT temps
jnh8	100	850	68	89	34	0s16	9	0s86
jnh9	100	850	95	173	179	0s75	8	0s71
dubois10	30	80	30	80	2063	0s15	641	1s30
dubois11	33	88	33	88	4111	0s31	1691	3s34
dubois12	36	96	36	96	8207	0s59	1835	4s48
dubois13	39	104	39	104	16399	1s18	3091	8s69
dubois14	42	112	42	112	32783	2s49	7693	14s39
dubois15	45	120	45	120	65551	5s15	9507	26s45
dubois16	48	128	48	128	131087	10s09	20733	65s97
dubois17	51	136	51	136	262159	19s69	23167	76s86

TAB. 8.1 – DP+ffis vs DP+TSAT : pour les instances de DIMACS

L'étendue de ces résultats montre très clairement que non seulement une méthode mixte de type DP+TSAT est tout à fait réalisable en pratique et que, par ailleurs, ce type d'approche est extrêmement efficace et dépasse de très loin en performance la procédure de Davis & Putnam classique.

8.6 Optimisations naturelles

Bien que les résultats précédents soient extrêmement positifs, il faut préciser que les expérimentations n'ont été conduites que dans l'objectif de vérifier la faisabilité d'une technique de recherche mixte. Du fait, aucune optimisation quant à l'implantation de DP+RL n'a été réalisée et de nombreuses pistes d'amélioration sont envisageables.

La première voie réside dans un bon paramétrage de la méthode. En effet, toute technique de recherche locale nécessite un réglage pointu de ses paramètres. Or, notre méthode repose sur une utilisation double de la recherche locale : dans le but de trouver un modèle et comme heuristique de branchement. Les paramètres spécifiques à chaque méthode de recherche locale, comme la probabilité de flipper une variable aléatoirement pour GSAT+RWS et pour WSAT+Noise ou la longueur de la liste tabou pour TSAT sont à notre avis importants mais non vitaux pour ce type de méthode. En effet, en choisissant une méthode de recherche locale pour laquelle on connait un réglage précis de ce paramètre, il est très simple de contourner ce problème. Par exemple, nous savons que la probabilité de 0.5 pour WSAT+Noise et pour GSAT+RWS est celle qui fournit les meilleures performances en moyenne pour ces méthodes de recherche locale et c'est donc cette valeur qui sera choisie lors de l'appel à une de ces procédures de recherche locale dans notre méthode mixte. Il nous semble qu'un paramètre plus crucial quant aux performances de notre algorithme réside dans les ressources (le nombre de flips et le nombre de tries) attribuées à la recherche locale. Plus précisément, si ces ressources sont insuffisantes, la méthode de recherche locale n'aura pas suffisamment de temps pour détecter d'éventuels noyaux inconsistants et pour se focaliser sur l'un d'entre eux. D'autre part si ces ressources sont trop abondantes, alors la méthode perd du temps dans la recherche locale qui, certes retournera certainement le « meilleur » littéral à affecter, mais ne pourra en aucun cas prouver seule l'inconsistance. Par conséquent, il nous semble crucial de déterminer le meilleur compromis entre le temps passé par DP et le temps passé par la RL. Plusieurs idées ont déjà été mises en pratique. Par exemple, les

ressources attribuées en termes de flips dépendent du nombre de variables dans le problème restant à traiter. Pour nos expérimentations ce nombre a été fixé à $20 \times V$ où V est le nombre de variables restant à affecter ; le nombre de tries étant pour sa part fixé à un. Cette manière de procéder nous semble un premier pas vers la mise au point du meilleur compromis puisque les ressources attribuées à la recherche locale dépendent ainsi clairement du nombre de variables non fixées par DP.

Une autre voie d'optimisation, est de limiter le nombre d'appels à la recherche locale. À chaque nœud (point de choix) de DP, un appel à la RL est effectué. À l'inverse dans l'algorithme `DPRL-pré` (cf. algorithme 8.3), un seul appel à la RL est effectué et le résultat de cet appel est exploité durant toute la recherche de DP. Entre ces deux points de vues extrêmes, nous pensons qu'une attitude optimale peut être trouvée en limitant le nombre d'appels aux méthodes de RL. Par exemple, pourquoi ne pas exploiter durant plusieurs nœuds successifs de l'arbre de DP les informations obtenues par un même appel à la recherche locale ? Ce compromis dépend, à notre avis, de la forme de la trace donnée par la recherche locale et de la profondeur de l'arbre de DP lors de l'appel à la RL.

Enfin, une dernière optimisation repose sur le constat que les meilleures heuristiques de branchement pour DP tentent d'équilibrer au maximum l'arbre de recherche. Or, dans l'exploitation que nous faisons des scores des littéraux, en aucune façon nous ne cherchons à équilibrer l'arbre. En effet, la méthode se contente de satisfaire le littéral étant apparu le plus souvent dans des clauses falsifiées lors de la RL. Nous pensons qu'une voie très prometteuse est d'utiliser ce score comme un facteur pondérant le poids des littéraux calculé de manière plus traditionnelle, c'est-à-dire en utilisant des critères syntaxiques (cf. paragraphe 7.1.3.2). En effet, une telle stratégie de branchement permettrait d'exploiter à la fois les informations sémantiques fournies par la RL et les informations syntaxiques du problème. Cette nouvelle heuristique de branchement devrait permettre d'étendre la portée de notre algorithme à un plus grand nombre d'instances de **SAT** , nous pensons particulièrement aux instances *k***SAT** aléatoires.

8.7 Conclusions

Les méthodes mixtes restent encore peu connues et peu utilisées. Cependant il nous semble, à la vue des résultats obtenus dans ce chapitre, que ces méthodes constituent une voie extrêmement prometteuse quant au traitement pratique d'instances de grande taille.

Notre contribution aux méthodes mixtes est multiple :

- Nous avons montré expérimentalement que les approches de RL jusqu'alors dédiées à la recherche de solutions, permettent de détecter et de localiser des noyaux inconsistants dans de larges bases de connaissances.
- Nous avons proposé de nouvelles approches complètes utilisant cette propriété pratique de localisation. Ces méthodes reposent toutes sur différents schémas de combinaisons entre DP et une méthode de recherche locale. Parmi elles, citons DPRL qui utilise la recherche locale à la fois comme un moyen de prolonger vers un modèle l'interprétation partielle construite par DP et comme heuristique de branchement pour DP.
- Les nombreuses expériences réalisées pour valider DPRL ont montré que non seulement cette approche était viable en pratique mais en outre qu'elle surpassait en performances (nombre de problèmes résolus et temps de calcul) les procédures de DP utilisant une heuristique syntaxique standard de branchement.
- Enfin, nous avons proposé nombre d'optimisations dans le but d'améliorer l'efficacité de DPRL.

Nous pensons, comme d'autres chercheurs [Selman *et al.* 1997], que les approches mixtes constituent

un véritable challenge pour l'avenir et devraient permettre la résolution d'instances intraitables actuellement.

Chapitre 9

Instances aléatoires 3SAT « déséquilibrées »

Nous présentons dans ce chapitre une série d'études que nous avons réalisées sur la génération d'instances aléatoires « déséquilibrées ». Dans le modèle standard (cf. paragraphe 5.2.1 page 46), la probabilité de tirer un littéral positif est de $\frac{1}{2}$. Nous avons étudié le comportement de ces instances en fonction de la probabilité de positiver un littéral. Ceci permet de faire varier la « proportion traitable » des instances générées et ainsi de couvrir un espace plus large de problèmes aléatoires.

Il est évident que déséquilibrer le nombre de littéraux positifs par rapport aux négatifs doit rendre les instances générées « plus faciles » à résoudre. C'est donc un peu à l'opposé des travaux de R. Génisson et de L. Saïs [Génisson & Saïs 1994] qui cherchent à rendre plus difficile le modèle standard en équilibrant les occurrences des littéraux dans les instances (cf. paragraphe intitulé *Génération d'instances* « *régulières* » page 50) que ces travaux s'inscrivent.

La première constatation réalisée est que la « complexité » des instances diminue inversement avec la probabilité de tirer un littéral positif. La courbe (cf. FIG. 9.1 page suivante) représente la complexité des instances lorsque l'on fait varier la probabilité (p) de 0.5 [1] à 0.9 toujours en fonction du rapport clauses / variables.

La complexité des instances est estimée en termes du nombre d'affectations de la procédure de Davis & Putnam associée à l'heuristique de Jeroslow & Wang [Davis *et al.* 1962, Jeroslow & Wang 1990] nécessaires à la résolution des instances. Pour chaque rapport C/V variant de 1 à 110 par pas successifs de 0.05 (V étant fixé à 100) et pour chaque probabilité p, 500 instances ont été testées.

Les pics de difficulté exhibés par ces courbes suggèrent la présence de seuils. La difficulté décroît inversement à p. En effet, l'augmentation de la probabilité permet de tendre vers la génération d'instances traitables (la probabilité de générer une clause de Horn est plus importante). Les courbes (cf. FIG. 9.2 page suivante) nous montrent les différents seuils observés.

Il faut également constater que plus la probabilité de générer des littéraux positifs augmente, plus les seuils se décalent vers la droite, c'est-à-dire plus il faut un nombre de clauses important pour obtenir des instances inconsistantes : pour 100 variables avec une probabilité de littéraux positifs de 0.9 il faut plus de

1. La courbe pour la valeur 0.5 correspond au modèle standard, elle est donnée a titre de référence.

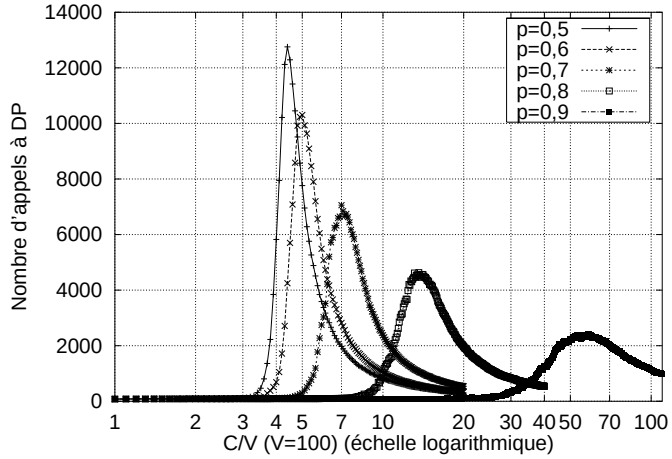

FIG. 9.1 – *Pic de difficulté pour des instances* **SAT** *aléatoires* « *déséquilibrées* »

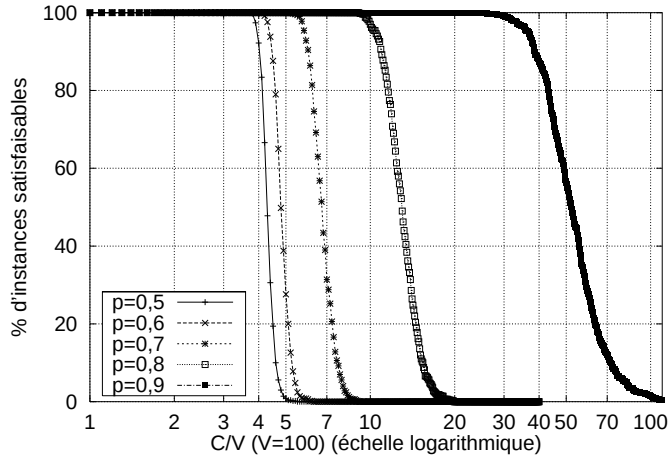

FIG. 9.2 – *Phénomènes de seuil pour les instances* **SAT** *aléatoires* « *déséquilibrées* »

10000 clauses pour commencer à obtenir une majorité d'instances non satisfaisables. A titre indicatif, les seuils observés sont donnés dans la table suivante (cf. TAB. 9.1). Cependant, afin d'obtenir des « valeurs de référence », il faudrait réitérer les expériences de nombreuses fois avec plus d'instances et en ajoutant une seule clause à chaque pas, au lieu de cinq ici. Malgré cela les variations par rapport aux nombres avancés dans cette table devraient être faibles.

p	0.5	0.6	0.7	0.8	0.9
C/V	4.25	4.80	6.75	12.95	52.10

TAB. 9.1 – *Valeurs expérimentales des seuils en fonction de* p

Par ailleurs, si l'on superpose les courbes des figures 9.1 et 9.2, nous constatons un phénomène surprenant : le pic de difficulté semble fonction de la proportion de littéraux positifs (cf. FIG. 9.3 page ci-contre) ! Pour le générateur d'instances aléatoires standard, il est connu que le pic de difficulté est atteint à 50% d'instances satsifaisables. Or, pour les instances aléatoires déséquilibrées ce pic n'est plus atteint à 50% d'instances satisfaisables mais respecte la proportion de littéraux négatifs. En effet, pour $p = 0.6$ correspondant à 60% de littéraux positifs, donc 40% de littéraux négatifs, le pic de difficulté est atteint aux

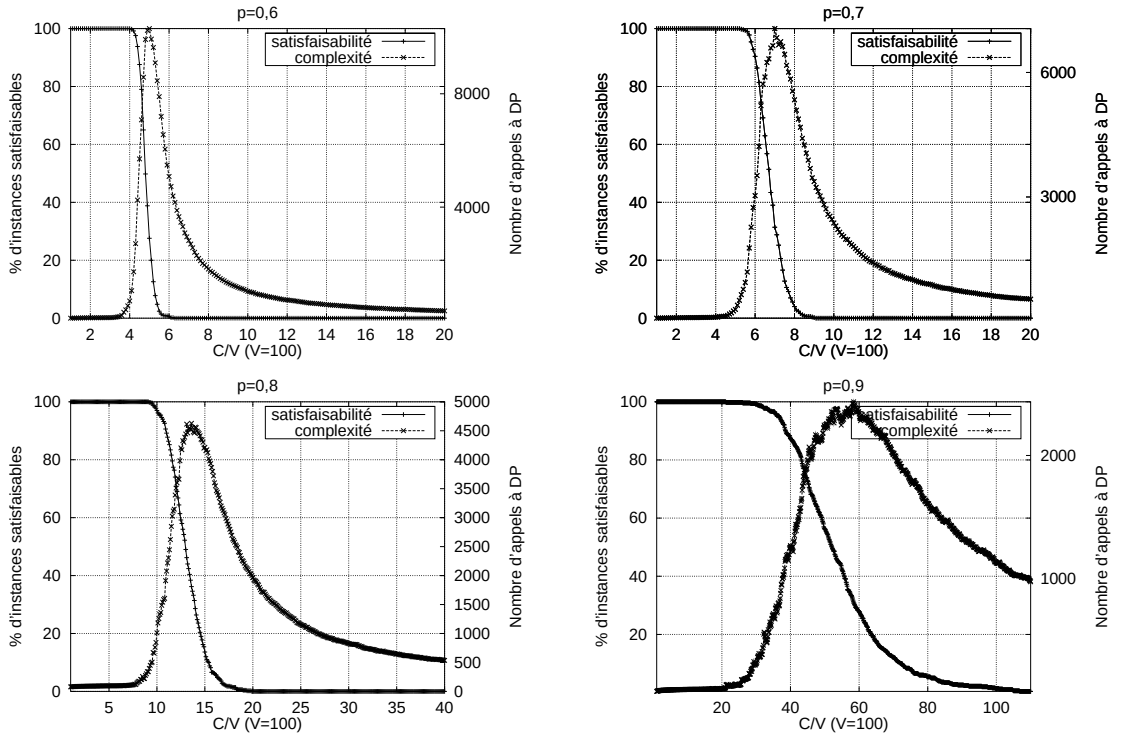

FIG. 9.3 – *Seuils et pics de difficultés pour les instances* **SAT** *aléatoires* « *déséquilibrées* »

alentours de 40% d'instances satisfaisables ! Ce phénomène se répète pour toutes les autres valeurs de p :

- $p = 0.7$: pic de difficulté localisé à environ 30% d'instances satisfaisables ;
- $p = 0.8$: pic de difficulté localisé à environ 20% d'instances satisfaisables ;
- $p = 0.9$: pic de difficulté localisé à environ 10% d'instances satisfaisables.

À notre connaissance, c'est la première fois qu'un tel phénomène est exhibé. Il reste à déterminer dans quelle mesure ce résultat peut être exploité pour l'amélioration des bornes théoriques du seuil et/ou pour l'élaboration de nouvelles stratégies ou heuristiques permettant une résolution plus efficace des instances aléatoires.

Par ailleurs, nous nous sommes intéressés aux littéraux unitaires et purs rencontrés lors de l'exécution de DP. Nous connaissons l'importance de ces littéraux pour cette méthode de résolution, puisqu'ils permettent de couper des branches de l'arbre de recherche de DP. Cependant, il nous semble qu'aucune étude sur le nombre d'apparitions de ces littéraux en fonction du rapport clauses sur variables n'a été réalisée. Cette étude, bien qu'empirique, pourrait aider à la compréhension de ces problèmes aléatoires. Parallèlement aux expériences précédentes, nous avons comptabilisé le nombre d'apparitions de littéraux unitaires (respectivement purs) au cours de la recherche de DP. Il faut noter que la version de Davis & Putnam que nous avons utilisée pour ces expériences affecte en priorité les littéraux unitaires, puis les littéraux purs et enfin les littéraux élus par l'heuristique de branchement[2], ceci afin d'élaguer au maximum l'arbre de recherche.

Les courbes (cf. FIG. 9.4 page suivante) présentent les résultats obtenus pour les littéraux unitaires en fonction de C/V et de la probabilité p de positiver une variable lors de la génération. Ces courbes sont très similaires à celles des pics de difficulté (cf. FIG. 9.1 page ci-contre), les extremums pour l'ensemble de ces

2. L'heuristique de branchement utilisée est celle proposée par Jeroslow & Wang [Jeroslow & Wang 1990]

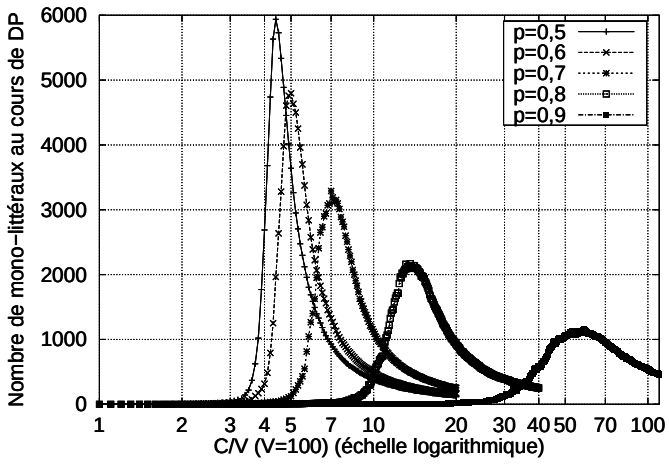

FIG. 9.4 – *Nombre de mono-littéraux lors de DP en fonction de C/V et de p*

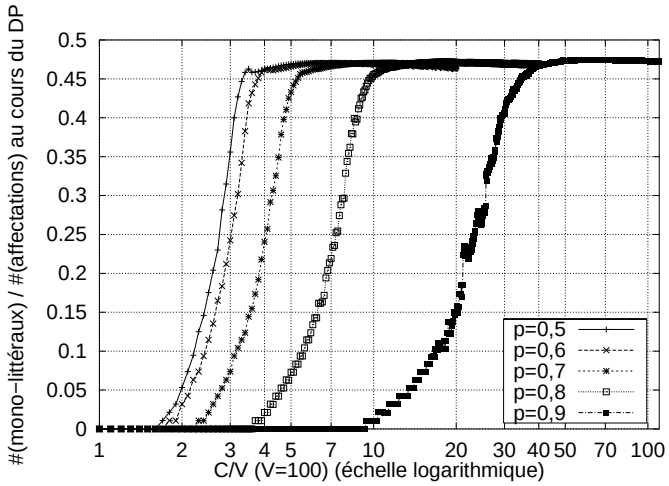

FIG. 9.5 – *Proportion de mono-littéraux par rapport au nombre d'affectations*

courbes au niveau du seuil. Cette constance s'explique aisément : plus la recherche est longue (ce qui est le cas au seuil) c'est-à-dire plus il y a d'affectations, plus la probabilité de produire des mono-littéraux en grand nombre est forte. Cette « propriété » peut se vérifier sur la courbe exprimant la proportion de mono-littéraux par rapport au nombre de littéraux affectés (cf. FIG. 9.5). À droite des différents seuils, c'est-à-dire là où la proportion d'instances satisfaisables avoisine les 100%, la proportion de mono-littéraux est quasiment nulle : pour exhiber un modèle, les littéraux unitaires ont un rôle plus secondaire. À l'approche du seuil, on constate une très rapide progression de la proportion de mono-littéraux : les instances deviennent plus contraintes, ce qui favorise l'application de la règle de propagation unitaire. Au seuil et à gauche du seuil : la proportion d'instances inconsistances est plus importante, et l'on constate que la proportion de littéraux unitaires par rapport aux nombre total d'affectations devient quasi-constante et ceci quelle que soit la probabilité p. Cette proportion n'évolue plus dès lors que le seuil est atteint, pour se situer entre 0.46 et 0.47 (quasiment une affectation sur deux est un mono-littéral) ! Ce qui confirme le rôle primordial de la propagation unitaire pour la résolution de problèmes difficiles. Ce résultat peut permettre la mise au point d'algorithmes efficaces [Li 1996, Li & Anbulagan 1997]. En outre le fait que la proportion se stabilise à partir du seuil permet ainsi d'obtenir un nouveau moyen de déterminer expérimentalement la valeur du seuil.

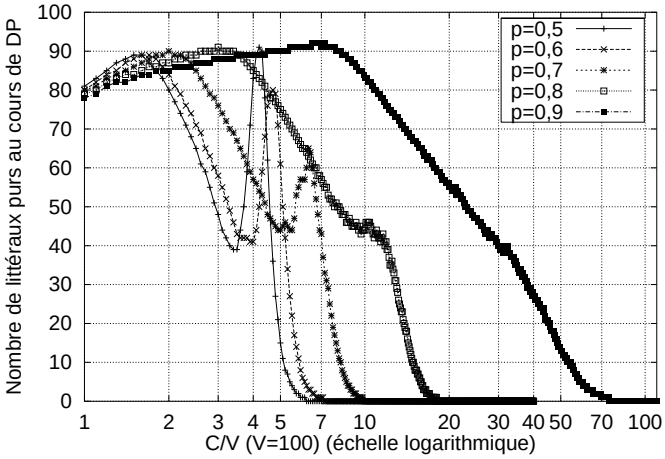

FIG. 9.6 – *Nombre de littéraux purs lors de DP en fonction de C/V et de p*

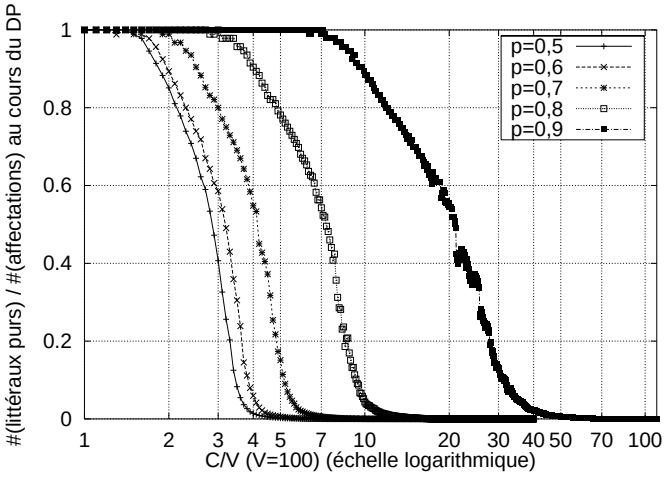

FIG. 9.7 – *Proportion de littéraux purs par rapport au nombre d'affectations*

Les courbes concernant les littéraux purs (cf. FIG. 9.6) présentent un tout autre aspect. En effet, ces courbes présentent deux extremums. Par exemple, la courbe $p = 0.5$ (correspondant au modèle standard) présente deux pics : le premier situé au rapport $C/V = 1.55$, le second au rapport 4.25 correspondant exactement à la valeur du seuil. Ces deux extremums, quelle que soit la probabilité p, se situent l'un à des valeurs très à gauche du seuil (dans une zone très peu contrainte) et l'autre exactement au seuil. Cette courbe à deux extremums, que nous avons appelée la courbe du chameau [3] reste relativement énigmatique : aucune idée intuitive ne permet d'expliquer le minimum local.

Par ailleurs, on constate que plus p est élévé, plus ce phénomène à « deux bosses » tend à disparaître et devient quasiment inexistant pour $p = 0.9$. À la vue des courbes du nombre de littéraux purs rencontrés au cours de l'arbre de résolution de DP, les courbes concernant la proportion de littéraux purs par rapport au nombre total de littéraux (cf. FIG. 9.7) affectés devraient être relativement surprenantes. En fait, la proportion de littéraux purs pour des rapports C/V avant les différents seuils est quasiment de 1 : toutes les affectations proviennent de littéraux purs. Ce phénomène s'explique par le faible nombre de clauses et les valeurs de p favorisant l'apparition de littéraux purs dès la génération du problème.

3. Uniquement en référence aux deux bosses dorsales de ce grand mammifère ruminant.

Cette proportion de littéraux purs devient constante et quasiment nulle à partir de la valeur du seuil.

En fait, nous constatons que dans la zone très peu contrainte, les littéraux purs permettent une accélération certaine de la recherche. Cependant, à partir du seuil, la règle des littéraux purs devient inutile.

Chapitre 10

Conclusion

Ce travail d'expérimentations et d'analyse des méthodes de résolution du problème **SAT** a abouti à plusieurs résultats originaux :

- Mise au point d'une technique incomplète à base de recherche locale tabou (TSAT) qui rivalise en performance avec les meilleures approches incomplètes actuelles.
- Mise en évidence d'un phénomène surprenant dans l'optimisation expérimentale de cette technique : linéarité de la courbe de la meilleure taille de liste tabou sur des instances *k***SAT** aléatoires.
- Introduction d'une nouvelle technique de génération de la configuration initiale basée sur le déséquilibre des variables.
- Mise au point d'une nouvelle technique d'élagage de l'arbre de DP basée sur une généralisation du théorème de partition du modèle.
- Mise en évidence des propriétés pratiques des méthodes de recherche locale dans la détection et la localisation de noyaux inconsistants dans de larges instances de **SAT** insatisfaisables.
- Mise au point de différentes techniques complètes basées sur différents schémas de combinaisons entre DP et la recherche locale (*e.g.* DPRL).
- Mise en évidence, lors de la validation expérimentale de DPRL, d'un gain de performance extrêmement important comparativement aux procédures de Davis & Putnam classiques.
- Mise en évidence, d'un lien entre le pic de difficulté des instances aléatoires et la probabilité de générer un littéral positif.
- Mise en évidence d'un phénomène surprenant quant aux nombres de littéraux purs rencontrés lors de la résolution des instances *k***SAT** aléatoires : la courbe du « chameau » exhibant deux maxima dont l'un est situé à la valeur du seuil.

Ces différents résultats ouvrent de nouvelles voies que nous souhaitons explorer.

Il est évident que l'une de nos préoccupations futures sera l'amélioration des différents algorithmes mis au point. Dans ce but, nous avons déjà proposé diverses optimisations, comme le réglage automatique de TSAT, l'introduction du concept de niveaux d'implications dans la procédure d'élagage de l'arbre de DP basée sur le théorème de partition du modèle généralisé restreint à la propagation unitaire, l'intégration d'outils syntaxiques dans la stratégie de branchements de DPRL, etc.

Par ailleurs, certains résultats expérimentaux obtenus ne pourront trouver explication sans une étude théorique approfondie. Nous pensons particulièrement :

- à la linéarité de la courbe de la taille optimale de la liste tabou pour les instances *k***SAT** aléatoires : dans quelle mesure ce phénomène peut-il être rattaché aux paysages des instances *k***SAT** aléatoires ?
- au lien entre le pic de difficulté des instances *k***SAT** aléatoires et la probabilité de générer un littéral positif ;
- à la courbe du « chameau » qu'aucune intuition ne permet d'expliquer.

Troisième partie

Compilation logique

Chapitre 11

Introduction – État de l'art

Représenter des connaissances et raisonner à partir de celles-ci est un problème central en Intelligence Artificielle. La connaissance est décrite classiquement dans un formalisme logique et plus particulièrement en logique propositionnelle pour le sujet qui nous intéresse. Cette base de connaissances est ensuite utilisée afin d'y extraire des informations contenues explicitement ou implicitement. Autrement dit, le problème abordé dans cette partie est celui de la déduction logique étant donné une base de connaissances exprimées en logique propositionnelle.

Ce chapitre introductif permet de situer notre étude aussi bien au travers des différents problèmes liés à la déduction logique qu'au travers des nombreuses approches déjà existantes. Le second chapitre de cette partie consacrée à la déduction logique et plus précisément à la compilation logique des bases de connaissances propositionnelles, présente notre contribution à ces domaines de recherche. Le dernier chapitre synthétise les résultats et les perspectives de nos travaux.

11.1 Problématique et définitions

Le problème de la déduction logique est un problème bien connu en intelligence artificielle. De manière générale, il consiste à savoir si une information donnée peut être déduite d'une base de connaissances exprimée dans un formalisme logique.

Notre étude est restreinte à la déduction logique à partir de bases de connaissances propositionnelles, et plus particulièrement à l'interrogation d'une CNF par une clause. Le problème traité dans cette partie se définit donc de la manière suivante :

Définition 11.1 (Interrogation d'une CNF par rapport à une clause)
Instance : Un ensemble de symboles propositionnels $\mathcal{S}$, Σ une formule de la logique propositionnelle construite sur $\mathcal{S}$ et mise sous forme normale conjonctive, c une clause construite sur $\mathcal{S}$.
Question : Est-ce que c est une conséquence logique de Σ : $\Sigma \models c$?

Il est clair que ce problème appartient à la classe de complexité coNP-complet. En effet, il revient à tester si la formule $(\Sigma \wedge \neg c)$ est inconsistante ($(\Sigma \models c) \equiv ((\Sigma \wedge \neg c) \models \bot)$), nous appelons **interrogation directe** cette manière de résoudre le problème de déduction logique. Le lien naturel existant entre satisfai-

sabilité et déduction explique que nombre de techniques développées pour effectuer la déduction logique sont empruntées à des approches initialement proposées pour résoudre **SAT**. Cependant, il est fréquent que de multiples déductions logiques soient effectuées avec la même base de connaissances. Par conséquent de nombreux appels au test de satisfaisabilité sont réalisés. Or au vu des performances actuelles des meilleures approches complètes pour **SAT**, la méthode d'interrogation directe devient très vite inexploitable en pratique.

Afin de pallier à cette non calculabilité pratique de la déduction logique en calcul propositionnel, plusieurs approches ont été proposées. Elles s'appuient sur différents principes ; en particulier :

- la ***restriction*** du langage utilisée, *e.g.* les clauses de Horn, ou de la relation de déduction employée, *e.g.* la résolution bornée ;
- l'***approximation*** de la base de connaissances ou de la relation de déduction ;
- la ***compilation logique*** de la base de connaissances.

Nous nous sommes particulièrement intéressés à ce dernier principe. L'idée est de déterminer une représentation $\widetilde{\Sigma}$ de la base initiale, qui se comporte comme Σ pour l'interrogation mais necessite un temps polynomial (en fonction de $|\widetilde{\Sigma}|$) pour ce faire. L'objectif de la compilation logique est donc de déterminer une représentation des connaissances (équivalente à Σ ou non) avec laquelle l'interrogation s'effectue efficacement (idéalement l'interrogation de $\widetilde{\Sigma}$ doit être garantie polynomiale en temps et en espace). Nous définissons donc la compilation logique d'une base de connaissances de la manière suivante :

Définition 11.2 (Compilation logique (exacte ou approchée))
Une **compilation logique**, *en vue de l'interrogation par une clause c, d'une CNF Σ est un triplet $\langle \Sigma, p_1, p_2 \rangle$ où p_1 est un algorithme de transformation tel que $p_1(\Sigma) = \widetilde{\Sigma}$ et où p_2 est un algorithme d'interrogation s'executant en temps polynomial tel que $p_2(\widetilde{\Sigma}, c) = \mathbb{V}$ si $\Sigma \vDash c$.*
On dit qu'une compilation logique est **exacte** *ou qu'elle préserve l'équivalence si : $p_2(\widetilde{\Sigma}, c) = \mathbb{V}$ si et seulement si $\Sigma \vDash c$.*
Une compilation logique est dite **approchée** *ou ne préservant pas l'équivalence si il existe une clause c telle que $p_2(\widetilde{\Sigma}, c)$ ne puisse pas être décidé (en temps polynomial).*

Pour une compilation logique exacte, tout en imposant aucune restriction sur la base de connaissances, il est impossible de garantir que la taille de la compilation ($|\widetilde{\Sigma}|$) est une fonction polynomiale de celle de Σ, à moins que $PH = \Sigma_2^p$ (effondrement de la hiérarchie polynomiale au second niveau !). Il est donc impossible de garantir l'efficacité d'une méthode de compilation en toute généralité. Cependant, l'objectif de la compilation étant d'obtenir une déduction plus efficace que l'interrogation directe, nous imposons la condition suivante :

Définition 11.3 (Compilable)
Une base de connaissances Σ est dite **compilable**, *si il existe une compilation logique $\langle \Sigma, p_1, p_2 \rangle$ et un entier naturel R à partir duquel pour tout entier n supérieur ou égal à R, n interrogations de Σ via la compilation sont plus efficaces que n interrogations directes.*

Autrement dit, si l'on note t la fonction qui retourne le temps d'exécution d'une opération, Σ est compilable si il existe une compilation logique $\langle \Sigma, p_1, p_2 \rangle$ et un entier naturel R tels que :

$$\forall\, n \geqslant R \;:\; \big(\, t(p_1(\Sigma)) + \sum_{i=1}^{n} t(p_2(\widetilde{\Sigma}, c_i)) \,\big) < \big(\, \sum_{i=1}^{n} t(\Sigma \vDash c_i) \,\big)$$

Si un tel entier R n'existe pas, la compilation n'est pas rentable et donc inutile d'où le terme de non compilable. Nous appelons **seuil de rentabilité** d'une compilation logique, cet entier R. Plus R est petit, plus la compilation est intéressante. Idéalement, il faudrait que $R = 1$ quelle que soit la clause testée.

Cependant, les temps d'interrogation (directe ou par compilation) ne dépendent pas uniquement de la base de connaissances, mais également des clauses testées[1]. Il est donc préférable de parler de temps moyen d'interrogation, ce qui nous donne pour R l'équation suivante, où t_m représente le temps moyen d'une interrogation et $[x]$ la partie entière de x :

$$R = \left[\frac{t(p_1(\Sigma))}{t_m(\Sigma \vDash c) - t_m(p_2(\tilde{\Sigma}, c))}\right] + 1$$

Ainsi, un des objectifs majeurs de la compilation logique de base de connaissances consiste à déterminer pour une base de connaissances donnée la compilation qui fournit la plus petite valeur de R. Par ailleurs, nous estimons que ce seuil de rentabilité est également un bon critère de comparaison des différentes méthodes de compilation logique.

11.2 État de l'art

Nous présentons très succinctement différentes approches basées sur la compilation logique. La liste des méthodes présentées dans cet état de l'art est loin d'être exhaustive ; pour une description plus complète et plus détaillée le lecteur pourra se référer à l'article [Cadoli & Donini 1999]. Notamment, nous passons sous silence une méthode nommée *vivification* de la connaissance [Levesque 1986, Borgida *et al.* 1989, Dalal 1995]. Très brièvement, la vivification est une méthode basée sur un raisonnement rendu traitable par l'utilisation conjointe de pré-traitements et de conclusions par défauts mais ceci aux dépends de la complétude.

Les méthodes de compilation peuvent être classifiées en deux grandes catégories. La première est celle comprenant les compilations qui préservent l'équivalence, c'est-à-dire celles qui construisent une représentation des connaissances équivalente à la base initiale. La seconde regroupe les compilations approchées qui permettent d'effectuer seulement, au travers d'une représentation non équivalente à la base de connaissances, une partie des déductions pouvant être faites depuis la base de connaissances initiale.

11.2.1 Les compilations préservant l'équivalence

Les méthodes de compilation exactes essayent de déterminer une représentation de toutes les informations, contenues explicitement ou implicitement dans la base de connaissances, afin que cette nouvelle représentation soit interrogeable en temps polynomial par rapport à la somme des tailles de la nouvelle représentation et de la question. Nous distinguons trois familles d'approches pour la compilation exacte :

- utilisation directe des impliquants ou des impliqués premiers ;
- ajout d'informations à la base de connaissance initiale, afin de pouvoir déduire n'importe quel fait par simple résolution unitaire ;
- utilisation d'impliqués premiers modulo une théorie.

1. Dans la troisième section de ce chapitre, nous discutons de cette difficulté à établir des « *benchmarks* » pour la déduction logique. En effet, contrairement à **SAT**, une instance est un couple : (base de connaissances, ensemble de clauses).

Nous proposons dans le chapitre 12 de nouvelles approches basées sur la notion de « couvertures traitables ». Cette notion permet de généraliser la dernière famille. Nous parlons alors de *compilations modulo des théories.* L'utilisation d'impliqués premiers modulo une théorie peut être vu comme une sous-famille de la famille des approches à base de compilations modulo des théories.

11.2.1.1 Compilation via les impliquants premiers et les impliqués premiers

Nous rappelons qu'un impliquant d'une base de connaissances Σ propositionnelle est un ensemble fondamental de littéraux I, assimilé à leur conjonction, tel que $I \models \Sigma$. En particulier les modèles d'une base de connaissances sont des impliquants. Par ailleurs, un impliquant I_p est dit premier pour Σ si pour tout autre impliquant I' de Σ, $I' \not\subset I_p$ (cf. définitions 2.38, 2.13 et 2.39).

Si l'on considère la disjonction de tous les impliquants premiers $I_{p_1}, I_{p_2}, \ldots, l_{p_k}$ de Σ, nous obtenons une formule DNF $I_{p_1} \vee I_{p_2} \vee \ldots \vee l_{p_k}$ qui est strictement équivalente à Σ. Ainsi quelle que soit la clause c,

$$\Sigma \models c \text{ si et seulement si } \forall\, i \in [1:k],\ I_{p_i} \models c\ .$$

I_{p_i} étant un monôme, vérifier si $I_{p_i} \models c$ est une opération élémentaire, réalisée en temps linéaire, consistant à s'assurer que l'intersection entre c et I_{p_i} n'est pas vide (auquel cas $I_{p_i} \models c$). Par conséquent, l'interrogation via la DNF s'effectue bien en temps polynomial par rapport à la taille de la DNF et à la taille de la clause testée.

Dualement, la conjonction de tous les impliqués premiers de Σ est équivalente à Σ (cf. les définitions des impliqués et impliqués premiers page 16). Si $C_{p_1} \wedge C_{p_2} \wedge \ldots \wedge C_{p_k}$ représente la CNF associée à l'ensemble des impliqués premiers de Σ, pour toute clause c :

$$\Sigma \models c \text{ si et seulement si } \exists\, i \in [1:k],\ C_{p_i} \models c\ .$$

C_{p_i} étant une clause, $C_{p_i} \models c$ revient à un test d'inclusion entre C_{p_i} et c ($C_{p_i} \models c$ si et seulement si $C_{p_i} \subseteq c$ ou c est une tautologie). Par conséquent, ici encore l'interrogation via l'ensemble des impliqués premiers est réalisable en temps polynomial par rapport à la taille cumulée de $(C_{p_1} \wedge C_{p_2} \wedge \ldots \wedge C_{p_k})$ et de c.

Intuitivement, l'ensemble des impliqués premiers d'une base de connaissances mise sous CNF peut être calculé en saturant par résolution la CNF : chaque résolvante étant un impliqué, il suffit de supprimer les impliqués non premiers (cf. principe de résolution page 53). Cependant, cette méthode requiert souvent de trop nombreuses résolutions pour être efficace en pratique. Le problème du calcul des impliqués premiers d'une CNF fait l'objet d'études depuis de nombreuses années, quelques algorithmes de calcul peuvent être trouvés dans [Tison 1967, Slagle *et al.* 1970, Reiter & De Kleer 1987, Jackson & Pais 1990, Madre & Coudert 1991, De Kleer 1992, Ngair 1993, Castell & Cayrol 1996b].

La plupart des techniques développées initalement pour le calcul d'impliqués (premiers) peuvent être adaptées au calcul des impliquants (premiers). Cependant des techniques spécialisées dans le calcul des impliquants et impliquants premiers ont également été développées [Dechter & Rish 1994, Schrag 1996]. Notamment, dans [Schrag 1996], Robert C. Schrag propose une méthode s'appuyant sur une variante de la procédure de Davis & Putnam. Pour une base de connaissances Σ, cette méthode consiste en une exploration complète de l'arbre de recherche de DP. Au niveau de chaque feuille aboutissant à un modèle, un impliquant premier est extrait de cet impliquant et est ensuite mémorisé. Il construit ainsi un ensemble d'impliquants premiers. Cet ensemble d'impliquants, vu comme leur disjonction, représente clairement une couverture d'impliquants premiers de Σ (cf. définition 2.41). Cet couverture est assurée irredondante par deux mécanismes :

1. la non exploration des sous-arbres ne pouvant aboutir qu'à des modèles déjà couverts par au moins un des impliquants premiers préalablement sauvegardés ;

2. la suppression, à chaque nouvel ajout d'un impliquant premier I_p, des impliquants I'_p tels que $I_p \subset I'_p$.

L'interrogation, via cette couverture d'impliquants premiers se fait à l'identique de celle pour la disjonction de tous les impliquants premiers.

11.2.1.2 Compilation par complétude de la résolution unitaire

L'idée dominante de ces méthodes de compilation logique exacte est d'ajouter à la base de connaissances suffisamment d'impliqués pour que la propagation unitaire devienne complète pour l'interrogation. Si l'on note Σ la base de connaissances, l'objectif de ces méthodes est de déterminer l'ensemble de clauses Ω tel que :

$$\forall\, imp \in \Omega,\ \Sigma \models imp \text{ et quelle que soit la clause } c\ ((\Sigma \wedge \Omega) \models^* c) \text{ ssi } (\Sigma \models c)^{2}$$

Alvaro Del Val propose dans [Del Val 1994] plusieurs algorithmes de production de l'ensemble d'impliqués Ω. Il est à noter que dans cette méthode l'interrogation s'effectue bien en temps polynomial : la propagation unitaire pouvant être réalisée en temps linéaire [Dowling & Gallier 1984].

Dans [Mathieu & Delahaye 1990, Mathieu & Delahaye 1994], Phillipe Mathieu et Jean-Paul Delahaye proposent une méthode similaire, appelée *achèvement*, afin de rendre le *chaînage avant*[3] complet pour la logique propositionnelle. Ces travaux sont complétés par ceux d'Olivier Roussel qui propose deux nouvelles méthodes d'achèvement : l'achèvement par parties et l'achèvement par cycles [Roussel & Mathieu 1996b, Roussel & Mathieu 1996a, Roussel 1997].

11.2.1.3 Compilation via des impliqués premiers modulo une théorie

Pierre Marquis dans [Marquis 1995b] généralise la notion d'impliqués et d'impliqués premiers et propose une méthode de compilation via cette généralisation.

Définition 11.4 (Impliqués et impliqués premiers modulo une théorie)
Un **impliqué modulo une théorie** Φ *d'une base de connaissances* Σ *est une clause* C *telle que* $\Sigma \cup \Phi \models C$, *que l'on note également* $\Sigma \models_\Phi C$.
C *est un* **impliqué premier modulo une théorie** Φ *de* Σ, *si pour toute autre clause* C' *telle que* $\Sigma \models_\Phi C'$ *et telle que* $C' \models_\Phi C$, *alors* $C \models_\Phi C'$.

Il faut noter que si l'on considère la théorie vide, nous retrouvons les définitions usuelles d'impliqués et d'impliqués premiers. Pierre Marquis suggère de choisir des théories telles que la déduction via cette théorie puisse être calculée en temps polynomial, c'est-à-dire qu'il existe un algorithme de décision pour $\Phi \models c$ (c étant une clause), tel que cet algorithme appartienne à P (*e.g.* les clauses de Horn, les clauses binaires, etc.). Ceci, afin de garantir la polynomialité de l'interrogation. Par ailleurs, dans [Marquis 1995a], il est montré que certaines restrictions de la déduction logique peuvent également être employées.

Ainsi, étant donné un ensemble $\mathcal{S}_\Phi$ d'impliqués premiers modulo une théorie Φ pour une base de connaissances Σ, nous avons quelle que soit la clause c :

$$\Sigma \models c \text{ si et seulement si } \exists\, C \in \mathcal{S}_\Phi,\ \text{tel que } C \models_\Phi c$$

2. Nous rappelons que nous notons $\models^*$ la déduction logique restreinte à la propagation unitaire (cf. défintion 7.3 page 115).
3. Le *chaînage avant* est un algorithme d'inférence fondé sur le modus ponens.

Il faut remarquer que $C \vDash_\Phi c$ si et seulement si pour chaque littéral l_i de C, $\Phi \vDash \neg l_i \vee c$. D'où le choix d'une théorie telle que la déduction pour cette théorie soit calculable en temps polynomial.

Dans [Marquis 1995b], sont donnés plusieurs exemples où le nombre d'impliqués premiers modulo une théorie est exponentiellement plus petit que le nombre d'impliqués premiers ajoutés dans la méthode proposée dans [Del Val 1994] (cf. section 11.2.1.2). Par ailleurs, dans [Marquis & Sadaoui 1996] est développé un algorithme de calcul des impliqués premiers de Σ modulo Φ qui ne requiert pas le calcul des impliqués premiers de Σ; ceci contraste avec nombre d'approches et rend envisageable la compilation de bases de connaissances qui n'étaient jusqu'alors pas compilables. L'algorithme proposé est fondé sur un diagramme de décision binaire (BDD « *Binary Decision Diagrams* ») et est une adaptation de celui développé dans [Madre & Coudert 1991]. Hormis une efficacité accrue comparativement à l'algorithme proposé dans [Marquis 1995b] et aux algorithmes fondés sur un raisonnement via les impliqués premiers « traditionnels », cet algorithme permet de traiter des formules qui ne sont pas nécessairement mises sous forme CNF.

11.2.2 Les compilations ne préservant pas l'équivalence

Une compilation logique approchée d'une base de connaissances Σ est une compilation pour laquelle l'interrogation via le résultat de la compilation n'est pas complète. La représentation A des connaissances obtenue par compilation est alors appelée approximation de Σ. Nous distinguons trois types de compilation logique ne préservant pas l'équivalence :

1. celles garantissant que si $A \vDash c$ alors $\Sigma \vDash c$;
2. celles garantissant que si $A \nvDash c$ alors $\Sigma \nvDash c$;
3. celles pour lesquelles la représentation des connaissances issue de la compilation se divise en deux approximations A et A' telles que si $A \vDash c$ alors $\Sigma \vDash c$ et si $A' \nvDash c$ alors $\Sigma \nvDash c$;

Pour ces différentes approximations, dans les cas non décrits, le résultat de l'interrogation est : « impossible de conclure ». L'approximation satisfaisant le premier critère est appelée couramment majorant de Σ, nous le notons Σ_{UB} (« *Upper Bound* »). Inversement l'approximation satisfaisant le second critère est appelée minorant et est notée Σ_{LB} (« *Lower Bound* »). Il faut remarquer que si $\Sigma_{UB} \equiv \Sigma_{LB}$, alors la compilation est exacte et préserve donc l'équivalence.

La plupart des méthodes de compilation logique approchée utilisent un « encadrement » de la base de connaissances par des ensembles de clauses de Horn. Cependant, une compilation approchée peut également être obtenue en arrêtant prématurément le calcul d'une compilation exacte.

11.2.2.1 Approximations issues de compilations exactes

La plupart des méthodes de compilation exacte discutées dans la section 11.2.1 peuvent être stoppées avant l'obtention d'une représentation des connaissances équivalente à Σ, sans pour autant que le calcul effectué soit inutile. La représentation ainsi obtenue peut dans certains cas être utilisée comme une approximation de Σ. On parle alors de méthodes de compilation « *anytime*[4] ».

En particulier, des méthodes de calcul de couvertures d'impliquants, comme la méthode proposée dans

4. La compilation peut être arrêtée à n'importe quel moment, par ailleurs plus l'arrêt est tardif, plus l'approximation est de bonne qualité.

[Schrag 1996], fournissent des minorants en cas d'arrêt prématuré : si une clause ne peut pas être impliquée par la couverture d'impliquants partiels, elle ne pourra pas l'être par la couverture entière.

Par ailleurs, des méthodes basées sur les impliqués comme celles proposées dans [Del Val 1994] ou dans [Marquis 1995b], si elles sont stoppées avant leur terminaison naturelle, fournissent des majorants de la base de connaissances initiale. Si la clause c par laquelle on désire interroger la base de connaissances Σ, est impliquée par un des impliqués de Σ déjà calculés, alors a fortiori $\Sigma \models c$.

11.2.2.2 Approximations de Horn

La méthode proposée dans [Selman & Kautz 1991] part de l'idée d'encadrer la base de connaissances Σ par deux ensembles de clauses Horn (Σ_{UB} et Σ_{LB}) tels que $\Sigma_{LB} \models \Sigma \models \Sigma_{UB}$. Cet encadrement permet ainsi de répondre affirmativement et négativement à une partie des interrogations faites à Σ, l'autre partie restera sans réponse. Par ailleurs, l'interrogation d'ensembles de clauses se calcule de manière linéaire, ce qui garantit la polynomialité de l'algorithme d'interrogation.

Il est naturel de rechercher le meilleur encadrement possible, c'est-à-dire de chercher deux ensembles de clauses de Horn Σ_{GLB} (« *Greatest Lower Bound* ») et Σ_{LUB} (« *Lowest Upper Bound* ») tels que :

$$\text{pour tout ensemble de clauses de Horn : } \Sigma' \left\{ \begin{array}{l} \text{si } (\Sigma_{GLB} \models \Sigma' \models \Sigma) \text{ alors } (\Sigma' \models \Sigma_{GLB}) \\ \text{si } (\Sigma \models \Sigma' \models \Sigma_{LUB}) \text{ alors } (\Sigma_{LUB} \models \Sigma) \end{array} \right.$$

Cependant, le calcul du meilleur encadrement ne peut, bien qu'il s'agisse d'une compilation approchée, être polynomial à moins que $P = NP$ [Selman & Kautz 1994]. Succinctement, Σ_{GLB} est obtenue en retirant certains littéraux des clauses non Horn de Σ de manière à obtenir des clauses de Horn. Σ_{LUB} est, quant à elle, constituée par un sous-ensemble des clauses de Horn qui sont des impliqués premiers de Σ. La taille de Σ_{LUB} peut être exponentielle en fonction de la taille de Σ. Ce problème peut être résolu dans certains cas en enrichissant le vocabulaire [Kautz & Selman 1992], cependant dans le pire cas, il est toujours impossible d'obtenir une borne Σ_{LUB} polynomiale, à moins que $NP \subseteq P/poly$ [Selman & Kautz 1994].

L'algorithme de calcul de Σ_{LUB} proposé dans [Selman & Kautz 1991] est en fait une saturation par résolution, avec la condition que chaque étape de résolution fasse intervenir au moins une clause non Horn. Alvaro Del Val propose d'imposer que chaque étape de résolution fasse intervenir exactement une clause non Horn [Del Val 1996]. Ce raffinement permet d'obtenir dans certains cas une borne supérieure exponentiellement plus petite qu'avec l'algorithme proposé dans [Selman & Kautz 1994]. Par ailleurs, récemment Yacine Boufkhad dans [Boufkhad 1998] a proposé de nouveaux algorithmes pour générer Σ_{GLB}, lesquels s'appliquent à de grandes bases de connaissances jusqu'alors intraitables. Dans cet article la notion de « plus grande borne inférieure » est étendue aux formules Horn renommables et une condition pour qu'une formule Horn renommable soit une Σ_{GLB} est établie.

Cet idée « d'encadrer » la base de connaissances a été étendue en utilisant d'autres types de déductions polynomiales [Selman & Kautz 1994, Del Val 1995] et a été généralisée au premier ordre [Del Val 1996]. Par ailleurs, différentes recherches ont été effectuées sur les propriétés calculatoires des compilations à base d'approximations de Horn [Cadoli 1993, Gogic *et al.* 1994].

11.3 Discussion

11.3.1 Objectifs

Nous présentons dans le chapitre suivant les différentes études que nous avons réalisées sur le problème de la compilation logique de bases de connaissances. Nous proposons notamment une extension aux impliquants du concept d'impliqués modulo une théorie et généralisons ce concept à la compilation modulo des théories [Mazure & Marquis 1996, Boufkhad *et al.* 1997].

Nous avons développé plusieurs approches de compilation autour de ces concepts. La plupart de ces algorithmes sont obtenus par adaptations d'algorithmes existants, comme celui proposé dans [Schrag 1996].

Par ailleurs, nous présentons diverses comparaisons et divers résultats expérimentaux pour ces techniques. Préalablement à ces différents tests, nous nous sommes confrontés au problème de la détermination d'instances d'évaluation pour ces algorithmes. En effet, vu le peu de « *benchmarks* » proposés dans la littérature pour le problème de la déduction logique, il n'était pas concevable de valider une approche sur ces seules instances. Nous nous sommes donc intéressés au problème de la détermination d'une base de tests.

11.3.2 Qu'est-ce qu'une instance difficile pour la déduction ?

Quel que soit le problème abordé (problèmes de décision, de recherche, d'optimisation, etc.), l'évaluation des algorithmes développés pour résoudre ces problèmes est à elle seule un problème. En effet, il faut d'abord établir des critères de comparaison pour ces algorithmes. Le temps CPU est considéré souvent comme le critère de jugement suprême, cependant il est tributaire de l'implantation des algorithmes et surtout de la puissance des ordinateurs sur lesquels sont testés les algorithmes. Suivant les algorithmes et les problèmes testés, des critères plus « équitables » peuvent être utilisés et pallier les problèmes liés au chronométrage (*e.g.* pour **SAT**, le nombre de nœuds de l'arbre de DP ou le nombre de réparations des méthodes de recherche locale). Cependant, il reste à déterminer sur quelles instances les algorithmes doivent être testés et évalués. Pour le problème **SAT**, il existe de nombreux « *benchmarks* » sur lesquels la communauté scientifique travaillant sur ce problème a pris l'habitude d'évaluer ses algorithmes (*e.g.* les instances proposées à différents challenges [DIM1993, BEI1996], les instances *k***SAT** aléatoires fournissant un panel illimité d'instances « difficiles », etc.). Cependant, un tel éventail d'instances n'est malheureusement pas disponible pour tous les problèmes. En l'occurrence, pour le problème de la déduction logique, il n'a pas été mis en place, à notre connaissance, de bases de données répertoriant un large ensemble d'instances pour ce problème.

Ainsi, l'évaluation des méthodes de compilation pour la déduction logique de bases de connaissances est difficile à réaliser faute d'instances clairement identifiées et caractérisées. Il est donc nécessaire d'emprunter des instances à d'autres problèmes pour lesquels les instances foisonnent. Le problème **SAT** étant fortement couplé au problème de la déduction, l'évaluation des méthodes de compilation logique s'effectue donc le plus souvent, sur des instances proposées initialement pour le test de consistance. Il est à noter qu'une première épuration est nécessaire afin d'exclure toutes les instances prouvées inconsistantes. Cependant, il reste à identifier les instances qui seront qualifiées difficiles pour la déduction logique. La difficulté d'une instance pour la déduction logique est mesurée en fonction de la difficulté d'une interrogation directe pour cette instance. Il nous faut donc déterminer des bases de connaissances pour lesquelles l'interrogation directe est jugée difficile ; ce sont d'ailleurs pour ces instances que la compilation logique est intéressante. C'est à ce niveau que l'on rencontre le plus de difficultés. En effet, une instance facile (ou difficile) pour **SAT** est-elle une base de connaissances facile (ou difficile) pour la déduction logique ?

Si la requête est réduite à la clause vide, la réponse à cette question est évidemment oui ; la déduction étant dans ce cas réduite au test de satisfaisabilité de l'instance elle même. Si la requête n'est pas la clause vide, la question reste en suspens. En effet, si une instance **SAT** est limitée à une base de connaissances, une instance pour la déduction logique est un couple : base de connaissances et requête(s). Nous montrons dans l'exemple ci-dessous, comment une instance dite « facile » pour **SAT** , peut être « difficile » pour l'interrogation directe d'une clause donnée.

Exemple 11.1 (Instance facile pour SAT et difficile pour la déduction)
Nous illustrons nos propos sur une instance bien connue de **SAT** : le problème des pigeons. Lorsque le nombre de pigeons est inférieur ou égal au nombre de pigeonniers, cette instance est satisfaisable et peut être résolue facilement par un DP classique (sans utilisation de symétries) même pour un nombre de pigeons très grand. À l'inverse, si le nombre de pigeons est supérieur au nombre de pigeonniers, cette instance est évidemment inconsistante et devient très vite, en pratique, hors de portée pour un DP n'utilisant pas les symétries.

Soit la base de connaissances CNF $\Pi_{(N,N)}$ représentant le problème des pigeons N–N (N pigeons pour N pigeonniers, cf. exemple 5.1 pour la représentation clausale du problème des 3 pigeons et 2 pigeonniers) et soit la question : y-a-t-il un pigeon dans le pigeonnier N ?

Cette question revient à savoir si de $\Pi_{(N,N)}$ il est possible de déduire la présence d'un pigeon dans le pigeonnier N, ce qui se traduit :

$$\Pi_{(N,N)} \models (p_{1,N} \vee p_{2,N} \vee \ldots \vee p_{N,N})$$ [5]

L'interrogation directe issue de cette déduction est réduite au test de satisfaisabilité suivant :

$$\Pi_{(N,N)} \wedge \neg p_{1,N} \wedge \neg p_{2,N} \wedge \ldots \wedge \neg p_{N,N} \models \bot$$

Or :

$$(\Pi_{(N,N)} \wedge \neg p_{1,N} \wedge \neg p_{2,N} \wedge \ldots \wedge \neg p_{N,N})^* = \Pi_{(N,N-1)}$$ [6]

Par conséquent, $\Pi_{(N,N)} \models (p_{1,N} \vee p_{2,N} \vee \ldots \vee p_{N,N})$ revient à tester la consistance de $\Pi_{(N,N-1)}$.

Le test de consistance de $\Pi_{(N,N-1)}$ étant une opération hors de portée pour la majorité des algorithmes de satisfaisabilité, nous avons montré que la déduction logique depuis une instance de **SAT** « facile » peut s'avérer « difficile ».

Par conséquent, l'identification de bases de connaissances difficiles pour la déduction logique est un problème complexe, auquel il est impossible de dissocier la clause à déduire.

Par ailleurs, nombre de techniques de compilation logique s'appuient sur un raisonnement sur les impliqués premiers ou les impliquants premiers. Nous savons que dans le pire cas, le nombre d'impliquants ou d'impliqués premiers d'une base de connaissances est exponentiel par rapport à la taille de la base de connaissances [Chandra & Markowsky 1978]. Ce travail théorique fut complété par les travaux empiriques de R. Schrag et J. Crawford sur le nombre d'impliqués et d'impliqués premiers des instances *k***SAT** aléatoires [Schrag & Crawford 1996]. Ils ont montré expérimentalement, que le pic du nombre d'impliqués (premiers) en fonction du rapport nombre de clauses (C) sur nombre de variables (V) correspond au pic de difficulté de ces instances, c'est-à-dire $C/V = 4,25$.

Vu la difficulté d'identifier des instances difficiles pour la déduction logique et vu les résultats obtenus sur les instances *k***SAT** aléatoires, nous avons testé les différentes techniques de compilation que nous avons

5. $p_{i,j}$ signifiant que le pigeon i se trouve dans le pigeonnier j.
6. Nous rappelons que Σ^* correspond à la base Σ simplifiée par la propagation unitaire.

développées sur ces instances *k***SAT** aléatoires. Nous utilisons pour valider ces méthodes des clauses de longueur fixe dont les littéraux sont générés aléatoirement parmi l'ensemble des littéraux constituant les bases de connaissances.

Chapitre 12

Compilations logiques modulo des théories

De nombreuses méthodes de compilation préservant l'équivalence sont fondées sur des mécanismes permettant de construire une représentation des connaissances à base d'impliqués et d'impliquants (premiers ou non) (cf. paragraphe 11.2.1.1). Citons notamment l'approche de de Reiter et De Kleer pour les impliqués [Reiter & De Kleer 1987] et l'approche de Schrag pour les impliquants [Schrag 1996]. Dans [Marquis 1995a, Marquis 1995b] est introduit le concept d'impliqués (premiers) modulo une théorie. Ce concept d'une part généralise celui d'impliqué et d'autre part a permis de compiler des bases de connaissances qui jusqu'alors n'étaient pas compilables [Marquis & Sadaoui 1996].

La compilation via les impliqués premiers modulo une théorie est basée sur le principe bien connu de « *diviser pour mieux régner* ». En effet, partant du constat qu'une base de connaissances Σ peut souvent être découpée en deux parties distinctes : une « facile » Φ et une « difficile » Ψ telles que $\Sigma \equiv (\Phi \wedge \Psi)$, P. Marquis propose de restreindre le calcul des impliqués de Σ au calcul des impliqués de Σ modulo Φ. L'idée centrale est d'utiliser autant que possible la partie facile dans le traitement de la partie difficile. À cette fin, Φ que l'on appelle également la théorie, sera choisie de telle manière que l'interrogation à partir de cette formule puisse être faite en temps polynomial. Un autre point remarquable réside dans le fait que cette partie facile peut être produite en utilisant les principes de restriction, approximation et autre compilation discutées au chapitre 11. Les compilations ainsi obtenues peuvent donc être génériques et par conséquent à la fois supplémentaires et complémentaires des autres approches.

Nous proposons dans un premier temps d'étendre ce principe d'impliqués modulo une théorie aux impliquants, c'est-à-dire de construire des compilations basées sur la notion d'impliquants modulo une théorie. Nous avons au mis au point une méthode de compilation reposant sur ce principe et la comparons avec la technique de compilation proposée par R. Schrag basée sur le calcul d'une couverture d'impliquants [Schrag 1996]. Nous généralisons par la suite ce concept à la compilation modulo des théories et proposons plusieurs techniques utilisant cette généralisation. Par ailleurs, nous montrons que les compilations par couverture d'impliquants ou d'impliquants modulo une théorie sont des cas particuliers des compilations modulo des théories. Après avoir présenté divers résultats expérimentaux obtenus par les différentes méthodes proposées, nous concluons et discutons de différentes perspectives à ce travail.

Ce travail est le fruit d'une collaboration avec Yacine Boufkhad, Éric Grégoire, Pierre Marquis et Lakhdar Saïs. Je les remercie d'avoir accepté que ce travail d'équipe figure à part entière dans mon mémoire de thèse.

12.1 Couvertures d'impliquants modulo une théorie

12.1.1 Définitions et notations préliminaires

Intuitivement, en raisonnement modulo une théorie, chaque déduction est remplacée par la déduction modulo une théorie.

Définition 12.1 (Conséquence logique modulo une théorie)
Étant données trois formules Φ, Σ *et* Ω *;* Σ *est* **une conséquence logique** *de* Ω **modulo la théorie** Φ, *appelée également* Φ**-conséquence** *et notée* $\Omega \models_\Phi \Sigma$, *si et seulement si* $\Omega \wedge \Phi \models \Sigma$.

Par extension, l'équivalence modulo une théorie se définit de la façon suivante :

Définition 12.2 (Équivalence logique modulo une théorie)
Étant données trois formules Φ, Σ *et* Ω *;* Σ *et* Ω *sont dites* **équivalentes modulo la théorie** Φ *ou* Φ**-équivalentes**, *notée* $\Omega \equiv_\Phi \Sigma$, *si et seulement si* $\Omega \models_\Phi \Sigma$ *et* $\Sigma \models_\Phi \Omega$.

Dualement à la définition d'impliqués (premiers) modulo une théorie (cf. définition 11.4), nous introduisons le concept d'impliquants (premiers) modulo une théorie :

Définition 12.3 (Impliquants et impliquants premiers modulo une théorie)
Un **impliquant modulo une théorie** Φ *ou un* Φ**-impliquant** *d'une base de connaissances* Σ *est un monôme* M *tel que* $M \models_\Phi \Sigma$.
M *est un* **impliquant premier modulo une théorie** Φ *ou* Φ**-impliquant premier** *de* Σ, *si pour tout autre monôme* M' *tel que* $M' \models_\Phi \Sigma$ *et tel que* $M' \models_\Phi M$, *alors* $M \models_\Phi M'$.

L'ensemble de ces concepts nous permet de définir la notion de couverture d'impliquants modulo une théorie.

Définition 12.4 (Couverture d'impliquants modulo une théorie)
Étant données deux formules Σ *et* Φ *; une* **couverture d'impliquants de** Σ **modulo la théorie** Φ *de* Σ, *notée* $\widehat{\Sigma}_\Phi$, *est un ensemble de* Φ*-impliquants de* Σ *(vus comme leur disjonction) tel que* Σ *est* Φ*-équivalent à* $\widehat{\Sigma}_\Phi$.

Clairement, ces nouvelles notions généralisent les concepts de conséquence logique, d'équivalence sémantique, d'impliquants et de couverture d'impliquants. En effet, il suffit de considérer Φ comme la théorie vide pour retrouver les définitions usuelles de ces concepts. D'autre part, trivialement nous avons :

Propriété 12.1
Toute couverture d'impliquants de Σ *est aussi une couverture d'impliquants modulo n'importe quelle théorie.*

Preuve (Propriété 12.1)
Pour toute couverture d'impliquants $\widehat{\Sigma}$ de Σ, nous avons $\widehat{\Sigma} \equiv \Sigma$, donc $\widehat{\Sigma} \models \Sigma$ et $\Sigma \models \widehat{\Sigma}$.
Or quelle que soit la formule Φ, $\widehat{\Sigma} \wedge \Phi \models \widehat{\Sigma}$, comme $\widehat{\Sigma} \models \Sigma$, nous avons $\widehat{\Sigma} \wedge \Phi \models \Sigma$, ce qui s'écrit encore $\widehat{\Sigma} \models_\Phi \Sigma$ (1).

De même, $\Sigma \wedge \Phi \models \Sigma \models \widehat{\Sigma}$, donc $\Sigma \models_\Phi \widehat{\Sigma}$ (2).
Nous obtenons d'après (1) et (2), $\forall \Phi,\ \widehat{\Sigma} \equiv_\Phi \Sigma$. □

Note 12.1
La réciproque est évidemment fausse, à savoir que toute couverture d'impliquants modulo une théorie de Σ n'est pas forcément équivalente à Σ, comme l'illustre l'exemple suivant.

Exemple 12.1 (Exemple d'équivalence modulo une théorie ne préservant pas l'équivalence usuelle)
Soient l'ensemble de clauses $\Sigma = \{(a \vee b), (\neg b \vee c), (\neg a \vee \neg c)\}$ et la théorie $\Phi = (b \wedge c)$.
Nous avons $(\neg a) \equiv_\Phi \Sigma$, mais $(\neg a) \not\equiv \Sigma$.

Propriété 12.2
Étant données trois formules Σ, Ψ et Φ, si $\Sigma \equiv (\Psi \wedge \Phi)$, alors $\Sigma \equiv (\widehat{\Psi}_\Phi \wedge \Phi)$.

Preuve (Propriété 12.2)
$\widehat{\Psi}_\Phi \equiv_\Phi \Psi$ (cf. définitiion 12.4), donc $(\Psi \wedge \Phi) \models \widehat{\Psi}_\Phi$ (1) et $(\widehat{\Psi}_\Phi \wedge \Phi) \models \Psi$ (2).
Si (1), alors a fortiori $(\Psi \wedge \Phi) \models (\widehat{\Psi}_\Phi \wedge \Phi)$. Cette dernière implication s'écrit également $\Sigma \models (\widehat{\Psi}_\Phi \wedge \Phi)$ (3) puisque Σ est équivalent à $(\Psi \wedge \Phi)$.
De la même manière, depuis (2), nous avons a fortiori $(\widehat{\Psi}_\Phi \wedge \Phi) \models (\Psi \wedge \Phi)$, se réécrivant $(\widehat{\Psi}_\Phi \wedge \Phi) \models \Sigma$ (4).
De (3) et (4), il découle que $\Sigma \equiv (\widehat{\Psi}_\Phi \wedge \Phi)$. □

Propriété 12.3
Étant données trois formules Σ, Ψ et Φ telles que $\Sigma \equiv (\Psi \wedge \Phi)$. Pour toute clause c, $\Sigma \models c$ si et seulement si $\forall \pi \in \widehat{\Psi}_\Phi,\ \pi \models_\Phi c$.

Preuve (Propriété 12.3)
$\Sigma \models c \Leftrightarrow (\widehat{\Psi}_\Phi \wedge \Phi) \models c$ (propriété 12.2) $\Leftrightarrow \widehat{\Psi}_\Phi \models_\Phi c$ (par définition) $\Leftrightarrow \forall \pi \in \widehat{\Psi}_\Phi,\ \pi \models_\Phi c$. □

Ce deux dernières propriétés permettent d'affirmer que si Φ est choisie de manière à ce que Φ soit reconnaissable et interrogeable en temps polynomial par rapport à la taille de Φ et de c, alors $\langle \widehat{\Psi}_\Phi, \Phi \rangle$ est une compilation logique de Σ préservant l'équivalence. Par ailleurs, il faut également que la classe polynomiale à laquelle appartienne Φ soit stable par adjonctions de clauses unitaires, ceci afin de garantir que $\pi \wedge \Phi$ reste interrogeable polynomialement. Ainsi par exemple, une formule appartenant à la classe polynomiale *Quad* (cf. paragraphe 3.3.6) ne pourra être retenue comme théorie Φ. Il reste que la plupart des classes polynomiales de **SAT** satisfont ces contraintes, particulièrement Horn, reverse-Horn, les clauses binaires, Horn-renommable, q-Horn, les clauses bien imbriquées, etc. (cf. paragraphe 3.3).

Les définitions suivantes formalisent nos propos et introduisent les notions de classes de formules traitables pour la compilation modulo une théorie et de compilation à base de couverture d'impliquants modulo une théorie.

Définition 12.5 (Classe traitable pour la compilation modulo une théorie)
Une classe C de formules propositionnelles est traitable pour la compilation modulo une théorie si :

- *il existe un algorithme (T_C) de temps polynomial permettant de tester l'appartenance d'une formule à C ;*
- *il existe un algorithme (Q_C) de temps polynomial permettant de vérifier si $\Phi \models c$ quelle que soit la formule Φ appartenant à C et quelle que soit la clause c (y compris la clause vide) ;*
- *pour toute formule Φ de C et pour tout littéral l, $(\Phi \wedge l) \in C$.*

Définition 12.6 (Compilation à base d'une couverture d'impliquants modulo une théorie)
Soient Σ, Ψ et Φ trois formules propositionnelles telles que $\Sigma \equiv (\Psi \wedge \Phi)$ et que Φ soit une formule appartenant à une classe C traitable pour la compilation modulo une théorie. Le triplet $\langle \Sigma, p_1, Q_C \rangle$ tel que $p_1(\Sigma) = \widehat{\Psi}_\Phi$ est une compilation de Σ préservant l'équivalence, appelé **compilation à base d'une couverture d'impliquants modulo une théorie**.

Nous proposons dans la prochaine section un algorithme permettant de calculer une couverture d'impliquants modulo une théorie. Nous discutons par ailleurs des différentes manières de diviser la base de connaissances Σ en deux formules Ψ et Φ telles que $\Sigma \equiv (\Psi \wedge \Phi)$.

12.1.2 Calcul des couvertures d'impliquants modulo une théorie

Dans ce paragraphe, nous présentons un algorithme simple (DPIC) [Mazure & Marquis 1996] pour calculer une couverture d'impliquants modulo une théorie. Nous proposons, par la suite diverses améliorations possibles de l'algorithme de base proposé.

12.1.2.1 L'algorithme de base

Notre objectif principal est de réduire la taille des couvertures d'impliquants obtenus lorsqu'un raisonnement modulo une théorie est appliqué. La proposition suivante montre comment cet objectif peut être atteint dans la plupart des situations. L'idée centrale consiste à simplifier la couverture d'impliquants de Ψ (la partie « difficile » de Σ) en écartant tout impliquant qui n'est pas maximal pour $\models_\Phi$.

Propriété 12.4
Étant données deux formules propositionnelles Ψ et Φ, si $\widehat{\Psi}$ est une couverture d'impliquants de Ψ, alors $\widehat{\Psi}_\Phi = \max(\widehat{\Psi}, \models_\Phi) = \{\pi \in \widehat{\Psi}$ t.q. $(\nexists\, \pi' (\neq \pi) \in \widehat{\Psi},$ t.q. $\pi' \models_\Phi \pi)\}$ est une couverture d'impliquants de Ψ modulo Φ.

Preuve (Propriété 12.4)
La propriété 12.1 énonce que $\widehat{\Psi}$ est également une couverture d'impliquants modulo n'importe quelle théorie Φ. Par ailleurs maximiser une couverture d'impliquants modulo une théorie Φ, ne fait que supprimer les informations redondantes de la couverture. Par conséquent $\max(\widehat{\Psi}, \models_\Phi)$ représente bien une couverture (irredondante) d'impliquants de Ψ modulo la théorie Φ. □

Cette proposition permet d'affirmer qu'il existe toujours moins (au pire le même nombre) d'impliquants dans $\widehat{\Psi}_\Phi$ que dans la couverture d'impliquants de Ψ correspondante.

Fort de cette dernière proposition, un algorithme basique pour la compilation modulo une théorie est dérivé de celui proposé dans [Schrag 1996]. Robert C. Schrag propose une méthode nommée DPPI pour calculer une couverture d'impliquants premiers d'une base de connaissances Σ. Cette méthode consiste en une exploration complète de l'espace de recherche en utilisant une procédure de Davis & Putnam. Au niveau de chaque feuille de l'arbre de DP aboutissant à un impliquant, un impliquant premier est extrait de cet impliquant et est ensuite mémorisé. À la fin de l'exploration, l'ensemble d'impliquants récoltés est clairement une couverture d'impliquants premiers de Σ.

Nous proposons d'étendre l'algorithme DPPI pour devenir DPIC et afin que ce dernier calcule non plus une couverture d'impliquants premiers de Σ mais une couverture d'impliquants premiers de Ψ modulo Φ telle que $\Sigma \equiv (\Psi \wedge \Phi)$. Volontairement nous conservons au maximum les noms de procédures proposés par R. Schrag dans [Schrag 1996] afin d'insister sur la similitude des deux approches.

Comme DPPI, DPIC admet en entrée une formule CNF Σ et retourne une formule DNF $\widehat{\Psi}_{\phi}$ structure équivalente à Σ pour laquelle l'interrogation est calculable en temps polynomial.

Algorithme 12.1 DPIC

```
1.  Procedure DPIC
2.  Input  : une formule CNF Σ ;
3.  Output : une représentation des connaissances équivalentes à Σ ;
4.  Begin
5.    Déterminer Ψ et Φ t.q. Σ ≡ (Ψ ∧ Φ) et Φ est traitable pour la compilation ;
6.    IC=∅ ;                                   %% IC («Implicant Cover» représente Ψ̂_Φ
7.    DP*(Σ, ∅) ;      %% DP* est un DP «classique» modifié pour récolter les impliquants
8.    return ⟨IC, Φ⟩ ;
9.  End

1.  Procedure DP*
2.  Input  : une formule CNF Σ et une interprétation partielle PI ;
3.  Output : modifie la variable globale IC (la couverture courante) ;
4.  Begin
5.    Σ=Propagation_Unitaire(Σ) ;                          %% cf. algorithme 7.2
6.    if (Σ contient la clause vide) then return ;
7.    elif (PI⊨ Ψ)
8.       PROCESS_IMPLICANT(PI) ;
9.       return ;
10.   else
11.      l=Heuristique_de_branchement(Σ) ;
12.      DP*((Σ ∧ (l)), PI ∪ {l});
13.      DP*((Σ ∧ (¬l)), PI ∪ {¬l});
14.   fi
15. End

1.  Procedure PROCESS_IMPLICANT
2.  Input  : une interprétation partielle PI ;
3.  Output : modifie la variable globale IC (la couverture courante) ;
4.  Begin
5.    if (∃ π ∈ IC t.q. PI ⊨_Φ π) then return ;
6.    Prime=ONE_PRIME(PI,Ψ) ;
7.    if (IC == ∅) et (Prime∧Φ consistant) then IC={Prime} ;
8.    else foreach π ∈ IC
9.          do
10.             if ((π ⊨_Φ Prime) then IC=IC/{π} ;
11.         done
12.         IC=IC∪{Prime} ;
13.   fi
14. End
```

Brièvement, DPIC décrit un arbre de décision complet pour Σ développé par la procédure `DP*` (une procédure à la Davis & Putnam). Lorsqu'un impliquant `PI` de Σ est trouvé, `PROCESS_IMPLICANT` vérifie si cet impliquant est maximal par rapport à $\models_\Phi$ pour la couverture d'impliquants courante. Si `PI` est maximal, un impliquant premier `Prime` de Ψ est extrait de `PI` de telle manière que `PI` $\models$ `Prime`. Ensuite, `PROCESS_IMPLICANT` compare `Prime` avec tous les autres impliquants de Ψ déja collectés et mémorisés dans `IC`, ceci afin de ne conserver que ceux qui sont maximaux par rapport à $\models_\Phi$ (un seul élément par classe d'équivalence modulo Φ). Enfin `Prime` est ajouté à `IC`.

`ONE_PRIME(PI,`Ψ`)` retourne un impliquant premier de Ψ tel que `PI` $\models \Psi$. Cette procédure considère successivement chaque littéral l de `PI` et vérifie si l est utile, c'est-à-dire s'il existe une clause c de Ψ telle que $c \cap$ `PI` $= \{l\}$. Si l est ainsi prouvé inutile, l est supprimé de `PI`. Plus de détails sur les techniques d'extraction d'impliquants premiers peuvent être trouvés notamment dans [Castell & Cayrol 1996b].

Les procédures `Propagation_Unitaire` et `Heuristique_de_branchement` sont des fonctions standard rencontrées dans toute implantation d'une procédure de Davis & Putnam. Ces fonctions ont fait l'objet de larges paragraphes dans le chapitre 7.

Il est assez aisé de s'assurer que DPIC calcule bien une couverture d'impliquants de Ψ modulo Φ équivalente à Σ. Cependant, il faut noter que ceci n'est pas dû à la propriété 12.4 puisque l'arbre de décision exploré par `DP*` n'est pas celui de Ψ mais celui de Σ. En conséquence, le développement d'une branche est stoppé dès que l'interprétation partielle `PI` associée à cette branche est un modèle de Σ (qui est a fortiori un modèle de Ψ) ; en effet, DPIC ne génère pas de couverture d'impliquants de Ψ.

Lorsque la théorie vide est choisie lors de la première instruction de DPIC (donc, $\Psi \equiv \Sigma$), DPIC est identique à l'algorithme DPPI proposé par Schrag. La seule différence[1], réside dans la procédure `PROCESS_IMPLICANT`. En effet chaque impliquant premier de Ψ généré par `ONE_PRIME` peut directement être ajouté à IC (s'il ne l'est pas déjà) puisque les impliquants premiers sont (par définition) maximaux par rapport à $\models$.

Il est évident qu'il est impossible de garantir la polynomialité de la taille de la couverture calculée par DPIC (en fonction de la taille de Σ). Une conséquence immédiate est que le temps requis par DPIC n'est pas non plus garanti polynomial par rapport à la taille de l'entrée. Il est à noter que, de ce point de vue, le comportement de notre approche n'est pas pire que celui des autres techniques préservant l'équivalence. En effet, d'un point de vue théorique, l'obtention d'une compilation de taille polynomiale pour n'importe quelle base de connaissances est impossible à moins que $NP \subseteq P/poly$. La preuve de cette inclusion aurait pour conséquence l'effondrement de la hiérarchie polynomiale au second niveau, ce qui est extrêmement improbable.

Par ailleurs, il est clair que le choix de Ψ et Φ peut grandement influencer les performances de l'algorithme en ce qui concerne notamment la taille de la compilation. Comme nous l'évoquions en introduction, différentes classes de théories traitables peuvent être utilisées. En fait, une « décomposition » peu judicieuse de Σ en Ψ et Φ peut être très pénalisante à la fois en ce qui concerne la taille de la compilation mais également en ce qui concerne le temps nécessaire à l'obtention de cette compilation.

Bien que l'arbre de décision exploré par DPIC contienne toujours moins de nœuds que celui exploré par DPPI (à heuristiques de branchement égales), nous ne pouvons garantir que le nombre d'éléments de la couverture d'impliquants premiers de Ψ modulo Φ générée par DPIC, soit toujours plus petit que le nombre d'éléments de la couverture d'impliquants premiers fournie par DPPI. Par ailleurs, nous ne pouvons pas garantir que le nombre de littéraux dans chaque impliquant premier calculé par DPIC est

1. La version présentée dans [Schrag 1996] n'inclut pas le test initial de la procédure `PROCESS_IMPLICANT`, l'absence de ce test ne permettait pas de garantir l'obtention d'une couverture irredondante d'impliquants premiers, cependant ce test a maintenant été ajouté à DPPI (communication personnelle).

inférieur à celui de l'impliquant premier correspondant généré par DPPI. Cependant, comme le montrent les résultats expérimentaux (cf. paragraphe 12.1.3), ces deux situations sont extrêmement rares en pratique. Dans le cas général, nous pouvons espérer raisonnablement que la couverture d'impliquants calculée par DPIC contienne moins d'impliquants (et de plus petites tailles) que celle obtenue à partir de DPPI. Les nombresuses expérimentations réalisés confirment cet état de fait.

12.1.2.2 L'algorithme d'interrogation

La proposition 12.3 et le choix d'une théorie Φ appartenant à une classe traitable pour la compilation modulo une théorie nous assurent qu'il est possible, étant donné le résultat de la compilation, de répondre à l'interrogation de n'importe quelle clause et ceci en temps polynomial par rapport aux tailles cumulées de la compilation et de la requête.

En fait, l'algorithme d'interrogation associé à DPIC est identique à celui proposé par R. Schrag pour DPPI. La seule différence réside dans la substitution de la déduction logique usuelle par la déduction logique modulo une théorie. Ainsi très simplement un algorithme d'interrogation de Σ à partir de forme compilée modulo une théorie s'écrit de la manière suivante :

Algorithme 12.2 DPIC-QUERY

```
1.  Function DPIC_QUERY : boolean
2.  Input  : ⟨Ψ̂_Φ, Φ⟩ obtenue à partir de DPIC(Σ) et une clause c à interroger ;
3.  Output : vrai si Σ ⊨ c, faux sinon ;
4.  Begin
5.    foreach π ∈ Ψ̂_Φ
6.    do
7.       if π ⊭_Φ c then return false ;
8.    done
9.    return true ;
10. End
```

Lorsque $\pi \vDash_\Phi c$ peut être décidée en temps linéaire par rapport à $|\Phi|$ (par exemple si Φ est complet pour la propagation unitaire), l'interrogation est calculée dans le pire cas, en $\mathcal{O}(\#\widehat{\Psi}_\Phi \times (|\pi_{\max}| + |c| + |\Phi|))$, où $\#\widehat{\Psi}_\Phi$ représente le nombre d'impliquants contenus dans la couverture et $|\pi_{max}|$ le nombre de littéraux du plus grand (en terme du nombre de littéraux) impliquant de $\#\widehat{\Psi}_\Phi$.

12.1.2.3 Améliorations possibles de DPIC

Nous discutons dans ce paragraphe de différentes variantes et améliorations pouvant être greffées à DPIC. Toutes ces variantes reposent sur différents compromis entre la taille de la couverture de Φ-impliquants et le temps nécéssaire à l'obtention de cette couverture.

Une première variante peut être obtenue en modifiant les conditions d'appel à `DP*`. En effet afin de collecter les Φ-impliquants de Ψ, DPIC utilise l'arbre de décision (décrit par `DP*`) développé pour la formule Σ. Or, `DP*` peut être appelé avec n'importe quelle formule Ω telle $\Sigma \vDash \Omega \vDash \Psi$, la couverture construite par `DP*` restant une couverture de Ψ modulo la théorie Φ. Particulièrement, `DP*` peut être appelé avec Ψ.

Dans l'algorithme de base, le choix de lancer la recherche sur Σ était motivé par le nombre de modèles de chacune des instances. En effet, il est évident que Σ admet toujours moins (au pire le même nombre) de modèles que n'importe quelle formule Ω, puisque $\Sigma \models \Omega$. Cependant ce choix peut être remis en question ; en effet, il est impossible d'affirmer que la couverture construite par DP*(Σ) est plus petite que celle de DP*(Ω).

Par ailleurs, le temps consommé par DP*(Ω) peut être sensiblement inférieur à celui de DP*(Σ). Particulièrement si l'on considère une formule Ω telle que son nombre de variables est strictement inférieur à celui de Σ, la recherche des Φ-impliquants de Ψ est grandement facilité. Par exemple, si l'on considère que Ψ et $(\Sigma \setminus \Psi)$ (=Φ) ne partagent pas de variables (partition du modèle) alors trouver un modèle de Ψ en parcourant l'arbre de recherche décrit pour Σ peut être aussi difficile que de trouver un modèle de Σ, alors que déterminer un modèle de Ψ en parcourant directement l'arbre de recherche décrit pour Ψ est nettement moins coûteux.

Une seconde amélioration concerne le test d'élagage (« *pruning test* ») effectué en première instruction de PROCESS_IMPLICANT. En fait, ce test peut être utilisé de manière plus « agressive ». En effet, chaque branche de l'arbre de recherche décrit par DP* peut directement être coupée si cette branche décrit une interprétation partielle qui n'est pas maximale pour $\models_\Phi$. Ainsi le test de maximalité effectué initialement dans PROCESS_IMPLICANT peut être déplacé au niveau de la première instruction de DP*. Bien évidemment la procédure PROCESS_IMPLICANT n'effectue plus ce test étant donné que chaque impliquant à traiter est issu d'une interprétation partielle maximale pour $\models_\Phi$. Il faut noter que l'arbre de recherche obtenu par cette optimisation est toujours plus petit que celui obtenu par l'algorithme de base. De plus, la taille de la couverture peut en conséquence être également réduite. En ce qui concerne le temps d'exécution de cette nouvelle version, rien ne garantit réellement que celui-ci soit inférieur au temps requis par la version basique de DPIC. En effet, ce test de maximalité étant relativement coûteux, il peut nuire aux performances de DP*.

Une troisième variante de DPIC peut être obtenu en en remplaçant la troisième instruction de DP* (ligne 7) par :

```
7 elif (PI ⊨Φ Ψ)
```

Une fois encore, l'arbre de décision décrit dans cette version de DPIC est toujours inférieur à celui exploré par la version basique de DPIC, sans pour autant nuire à la validité de l'algorithme. Dans cette version, la procédure ONE_PRIME doit être modifiée afin de vérifier successivement si chaque littéral de PI est nécessaire pour satisfaire Ψ étant donné Φ (si ce n'est pas le cas, le littéral est supprimé de PI). Par conséquent, le Φ-impliquant de Ψ produit par ONE_PRIME n'est plus un impliquant premier de Ψ lequel est souvent plus long (en terme du nombre de littéraux) que le Φ-impliquant.

Une autre variante, apellée « mixed », de DPIC particulièrement intéressante consiste à vérifier lorsque PI est un impliquant de Ψ, si cet impliquant n'est pas également un impliquant de Σ (test extrêmement rapide). Ainsi chaque impliquant premier extrait de PI est marqué de telle manière à savoir s'il est un impliquant de Σ ou un impliquant de Ψ. La couverture sauvegardée reste identique à celle contruite dans la version basique de DPIC. Cependant, nous obtenons ainsi un moyen de savoir si l'impliquant mémorisé est un impliquant de Σ ou un impliquant de Ψ. Ce marquage permet de modifier l'algorithme d'interrogation (cf. algorithme 12.3) de manière à ce que celle-ci se fasse le plus efficacement possible. En effet, si un impliquant de la couverture est un impliquant de Σ, un test d'inclusion (effectué plus efficacement qu'un test d'implication modulo une théorie) correspondant au test d'implication usuelle est suffisant.

Algorithme 12.3 DPIC-QUERY-MIXED

```
1.  Function DPIC_QUERY_MIXED : boolean
2.  Input  : ⟨Ψ̂_Φ, Φ⟩ obtenue à partir de DPIC(Σ) et une clause c à interroger ;
3.  Output : vrai si Σ ⊨ c, faux sinon ;
4.  Begin
5.    foreach π ∈ Ψ̂_Φ
6.    do
7.       if (π est marqué) then
8.         if (π ∩ c = ∅) then return false ;          %% équivalent à si π ⊭ c
9.       else
10.        if (π ⊭_Φ c) then return false ;
11.      fi
12.   done
13.   return true ;
14. End
```

Ceci permet d'accélérer de façon notable l'algorithme d'interrogation sans nuire aux performances et à la validité de DPIC.

Enfin, il faut noter que DPIC (ou les variantes discutées ci-dessus) sont « *anytime* ». Plus précisément, lorsque DPIC est arrêté avant sa terminaison naturelle, l'ensemble IC de Φ-impliquants collecté peut être considéré comme un minorant d'une couverture d'impliquants de Ψ module Φ, comme discuté dans le paragraphe 11.2.2.1. C'est-à-dire qu'à chaque interrogation par une clause c, si $\mathtt{IC} \not\models_\Phi c$, alors il est possible de conclure que $\Sigma \not\models c$. Par ailleurs, plus les ressources de temps attribuées à DPIC sont grandes, plus le minorant sera précis, jusqu'à l'obtention d'une compilation exacte.

12.1.3 Résultats expérimentaux

Nous présentons dans cette section divers résultats expérimentaux obtenus par notre approche. Nous nous sommes particulièrement intéressés à la question de savoir quels étaient les avantages computationnels des approches à base de couverture d'impliquants modulo une théorie par rapport aux approches basées sur une couverture classique d'impliquants.

Comme discuté dans le paragraphe 11.3.2, nos expérimentations ont porté sur des instances aléatoires. En particulier nous avons comparé l'algorithme DPPI [Schrag 1996], reconnu d'un point de vue pratique comme l'une des meilleures approches à base de couverture d'impliquants avec DPIC utilisant la variante dite « *mixed* » (DPIC+mixed) décrite dans le paragraphe précédent.

Ces algorithmes ont été comparés sur trois critères temporels :

1° le temps de compilation ;
2° le temps d'interrogation ;
3° le temps cumulé (compilation et interrogation).

et sur un critère spatial : la taille de la compilation. En ce qui concerne l'approche de R. Schrag, la taille de la compilation correspond exactement au nombre de littéraux contenus dans la couverture calculée, pour

notre approche cette taille correspond au nombre de littéraux dans la couverture additionné à la taille de Φ (taille exprimée également en nombre de littéraux).

Avant de mener toute expérimentation, il fallait déterminer quelle stratégie adopter pour scinder Σ en deux formules Ψ et Φ. Nous avons choisi de prendre pour Φ l'ensemble de clauses de Horn issu de Σ et pour Ψ l'ensemble de clauses correpondant à $(\Sigma \setminus \Phi)$, il faut noter que Ψ correspond à un sous-ensemble des clauses reverse-Horn de Σ, car Σ est une intance *3***SAT**. Par ailleurs, Φ appartient bien à une classe traitable pour la compilation modulo une théorie (**HORN-SAT**).

Les expérimentations ont été menées sur des instances *3***SAT** aléatoires de 100 variables pour un rapport nombre de clauses sur nombre de variables de $4,15$ [2]. Par ailleurs, pour chacune des instances traitées 50000 questions, représentées par des clauses de longueur 3, ont été générées aléatoirement en utilisant un alphabet identique à celui utilisé lors de la génération des instances. Nous avons étudié le pourcentage d'instances traitées (sur un échantillon de 300 instances) en limitant les différents critères ci-dessus.

La première courbe montre le pourcentage d'instances compilées en fonction de différentes limitations de temps.

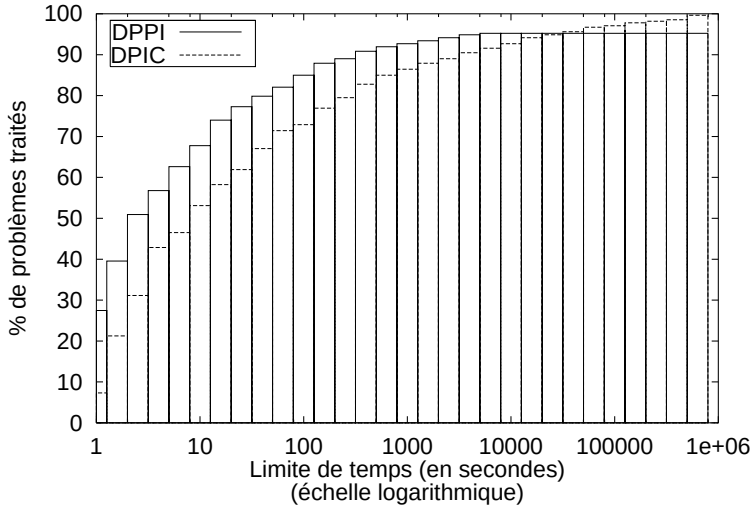

FIG. 12.1 – *Pourcentage d'instances 3***SAT** *compilées en fonction d'un temps limite*

En moyenne, pour une limitation de temps donné, DPPI permet de compiler plus d'instances que notre approche, cependant l'écart s'amenuise au fur et à mesure que les limitations de temps augmentent jusqu'à ce que cet écart soit réduit à zéro et devienne favorable à DPIC. Pour le temps limite maximum, DPIC a réussi à compiler 100% des instances, tandis que DDPI n'a pu traiter que 95% des instances.

La courbe suivante exprime également une contrainte de temps mais cette fois-ci sur le temps d'interrogation de la base par les 50000 clauses (requêtes).

Cette courbe montre que malgé une procédure d'interrogation plus complexe, DPIC-QUERY-MIXED est plus rapide que DPPI-QUERY. Ceci est dû notamment au fait que la taille des couvertures obtenues par DPPI est beaucoup plus petite que celles obtenues par DPPI (cf. FIG. 12.4).

Pour finir avec les limitations de temps, nous avons maintenant limité le temps cumulé de compilation et

2. Le seuil n'a pas été retenu ici car le nombre d'impliquants des instances au seuil est nettement inférieur à celui des instances situées à gauche du seuil, c'est-à-dire dans la partie sous-constrainte.

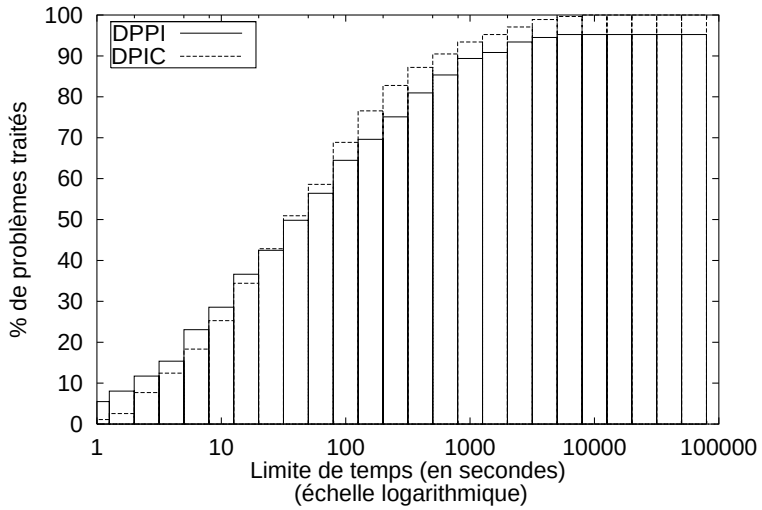

FIG. 12.2 – *Pourcentage d'instances 3***SAT** *interrogées en fonction d'un temps limite*

d'interrogation pour chacune des méthodes. La courbe ci-dessous montre que les pourcentages d'instances traitées globalement (compilation et interrogation) pour une limitation de temps donnée et pour chacune des méthodes sont comparables. Par ailleurs, étant donné que l'interrogation est en moyenne plus efficace avec DPIC, il existe donc un nombre d'interrogations pour lequel la compilation via DPIC sera toujours plus intéressante que celle utilisant DPPI.

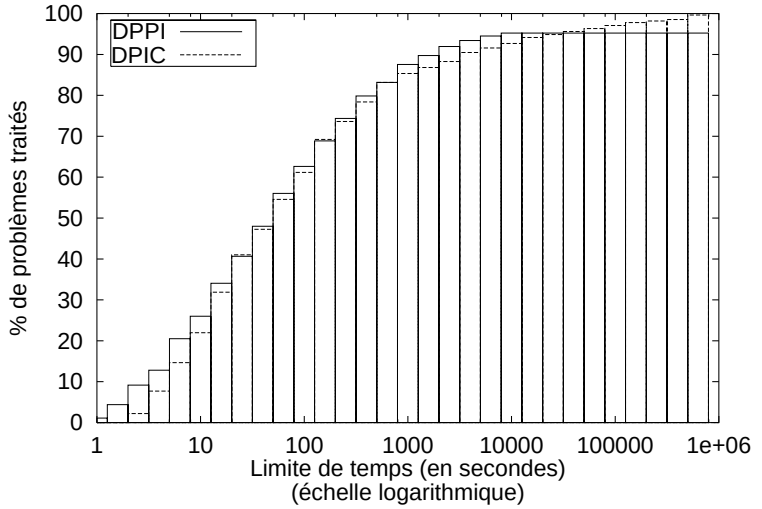

FIG. 12.3 – *Pourcentage d'instances 3***SAT** *compilées et interrogées en fonction d'un temps limite*

Magré les bons résultats obtenus par DPIC d'un point de vue temporel, il faut remarquer que l'atout majeur de cette méthode réside dans la taille des couvertures construites. En effet, si l'on limite la taille (en terme de littéraux) des couvertures calculées par chacune des méthodes, on s'aperçoit que quelle que soit la limitation de taille choisie, DPIC produit une couverture de taille strictement inférieure à celle de DPPI. Les résultats obtenus par les deux méhodes pour chaque limitation de taille sont synthétisés dans la figure 12.4. Il est à noter que ceci explique l'efficacité de DPIC par rapport à DPPI lors des phases d'interrogation : le nombre d'impliquants à tester pour DPIC est toujours inférieur à celui de DPPI.

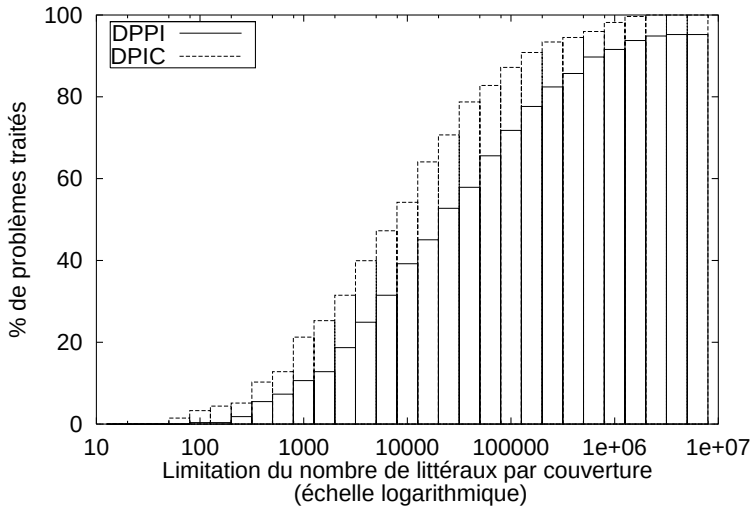

FIG. 12.4 – *Pourcentage d'instances 3***SAT** *compilées en fonction d'un espace mémoire limité*

12.1.4 Conclusion

Nous avons introduit dans cette section le concept de compilation modulo une théorie. L'exploitation faite de ce concept pour la compilation est le calcul de couverture d'impliquants modulo une théorie. Ce type de couverture généralise, la couverture d'impliquants usuelle, qui peut alors être vue comme une couverture d'impliquants modulo la théorie vide.

Par ailleurs, nous avons développé une approche (DPIC) de compilation préservant l'équivalence à base d'une couverture d'impliquants modulo une théorie. Cette nouvelle technique de compilation est en pratique tout a fait réalisable et, de surcroît, permet d'obtenir de meilleures performances notamment en ce qui concerne la taille de la compilation. Par ailleurs, cette approche est générique dans le sens où diverses classes traitables peuvent être choisies pour Φ. De plus, DPIC admet un certain nombre de variantes permettant de réduire soit le temps de compilation, soit la taille de la compilation. Cette approche comme la plupart des méthodes de compilation basées sur la collecte d'impliquants est « *anytime* ».

12.2 Couvertures d'impliquants modulo des théories

Les compilations modulo une théorie sont restreintes à l'utilisation d'une théorie traitable. Nous proposons de généraliser cette approche. À cet effet, nous présentons dans ce paragraphe une nouvelle technique de compilation logique préservant l'équivalence basée sur le concept de couvertures traitables (couvertures modulo des théories).

Sommairement, une couverture traitable d'une base de connaissances Σ est un ensemble fini $\mathcal{T}$ de formules traitables Φ (vues comme leur disjonction) tel que $\Sigma \equiv \mathcal{T}$. L'idée est donc de déterminer des théories traitables Φ couvrant des sous-ensembles de modèles de Σ les plus grands possibles, ceci afin de limiter la taille de $\mathcal{T}$.

Contrairement aux compilations modulo une théorie, les compilations à base de couvertures traitables tirent avantages de multiples classes polynomiales simultanément. De ce point de vue, elles sont donc une

généralisation du principe de compilation modulo une théorie et a fortiori de la technique à base de couverture d'impliquants proposée dans [Schrag 1996]. En outre, nous montrons que les couvertures traitables sont calculées et volontairement représentées au travers d'interprétations partielles.

Après avoir formellement défini la notion de couverture traitable, nous en étudions deux cas particuliers. Dans le premier cas, les interprétations partielles sont utilisées pour extraire des formules traitables de la base de connaissances. Dans le second cas, elles permettent d'obtenir des formules Horn-renommables. Nous concluons après avoir présenté quelques résultats expérimentaux.

12.2.1 Définitions et notations préliminaires

Précédemment dans ce chapitre, nous avons défini ce qu'était une classe traitable pour la compilation modulo une théorie (cf. définition 12.5 page 175). L'idée de compilation modulo des théories repose sur l'utilisation simultanée de plusieurs classes traitables. Nous étendons donc la définition précédente afin de préciser quelles sont les classes traitables considérées.

Définition 12.7 (Ensemble de classes traitables pour la compilation modulo des théories)
Une ensemble Cs de classes traitables pour la compilation modulo des théories est tel que pour chaque classe traitable C de Cs :

- *il existe un algorithme (T_C) de temps polynomial permettant de tester l'appartenance d'une formule à C ;*
- *il existe un algorithme (Q_C) de temps polynomial permettant de vérifier si $\Phi \models c$ quelle que soit la formule Φ appartenant à C et quelle que soit la clause c (y compris la clause vide) ;*
- *pour toute formule Φ de C et pour tout littéral l, il existe une classe traitable C' appartenant à Cs telle que $(\Phi \wedge l) \in C'$.*

Nous notons T_{Cs} l'algorithme qui, pour une formule Φ, vérifie si elle appartient à une des classes de Cs et nous désignons par Q_{Cs} l'algorithme qui, pour tout Φ appartenant à une des classes de Cs et pour toute clause c, vérifie si $\Phi \models c$.

Remarque 12.1

1. Il est clair à la vue de cette définition que toute classe traitable pour la compilation modulo une théorie peut appartenir à un ensemble de classes traitables pour la compilation modulo des théories. En outre, si C est une classe appartenant à Cs telle que C est traitable pour la compilation modulo une théorie, alors la classe traitable C' introduite dans le troisième alinéa de la définition est telle que $C' = C$.
2. Clairement les algorithmes T_{Cs} et Q_{Cs} sont polynomiaux en temps par rapport à la taille de la formule à tester et à la taille de Cs.

Comme précédemment, une grande majorité des classes de formules traitables pour **SAT** (ensemble de clauses de Horn, ensemble de clauses binaires, ensembles de clauses Horn-renommable, ensemble de clauses q-Horn, ensemble de clauses bien imbriqués, etc.) sont également traitables pour la compilation modulo des théories.

Définition 12.8 (Couverture tratable)
Soient une formule CNF Σ et un ensemble fini Cs de classes traitables pour la compilation modulo une théorie. Une **couverture traitable** *de Σ modulo Cs est un ensemble fini $\mathcal{T}$ de formules satisfaisables (vues comme leur disjonction) tel que :*

- $\forall\ \Phi \in \mathcal{T}\ :\ \exists\ C \in Cs$ t.q. $\Phi \in C$;
- $\mathcal{T} \equiv \Sigma$.

Clairement, la notion de couverture traitable généralise celle de couverture (d'impliquants) modulo une théorie Φ (cf. définition 12.4). En effet, cette dernière est un cas particulier de compilation modulo des théories où l'ensemble Cs est réduite à la seule classe traitable dont Φ est issue.

Dans le reste du manuscrit, nous supposons que Cs contient toujours au moins la classe $\{m$ t.q. m est un monôme$\}$[3]. Cette hypothèse assure qu'il existe toujours au moins une couverture traitable pour Σ : celle des impliquants premiers.

Nous nous sommes particulièrement intéressés aux couvertures traitables qui, comme pour les couvertures d'impliquants (premiers) classiques ou modulo une théorie, peuvent se représenter sous la forme d'une disjonction d'interprétations partielles. Ce choix est motivé par un souci de compacité de la représentation de la couverture traitable. Il reste qu'à partir de cette représentation, il doit toujours être possible de retrouver l'ensemble $\mathcal{T}$ en un temps polynomial. Dans cette optique, nous proposons deux nouveaux[4] cas particuliers de couvertures traitables :

1° les couvertures de simplifications traitables ;
2° les couvertures d'hyper-impliquants.

12.2.2 Les couvertures de simplifications traitables

Une première voie pour dériver à partir d'un ensemble de clauses Σ des formules traitables, est de simplifier Σ par une interprétation partielle. Notre objectif est donc de déterminer une interprétation partielle $I = \{l_1, l_2, \ldots, l_n\}$ telle que la formule $\Sigma_{l_{[1:n]}}$ soit traitable.

Définition 12.9 (Couvertures à base de simplifications traitables)
Soit Σ une base de connaissances et Cs un ensemble fini de classes traitables pour la compilation modulo des théories :

- *Une* **simplification traitable** *de Σ pour Cs est une interprétation partielle $\{l_1, l_2, \ldots, l_n\}$ telle qu'il existe une classe traitable C dans Cs telle que $\Sigma_{l_{[1:n]}}$ soit satisfaisable et appartienne à C.*
- *Une* **couverture de simplifications traitables** *de Σ pour Cs est un ensemble $\widehat{\Sigma}_{[]}$ de simplifications traitables de Σ pour Cs (vues comme leur disjonction) tel que $(\widehat{\Sigma}_{[]} \wedge \Sigma) \equiv \Sigma$.*
- *Une* **compilation à base d'une couverture de simplifications traitables** *est le triplet $\langle \Sigma, p_1, Q_{Cs} \rangle$ tel que $p_1(\Sigma) = \widehat{\Sigma}_{[]}$.*

La proposition suivante nous assure qu'une compilation modulo une couverture de simplifications traitables préserve l'équivalence.

Propriété 12.5
Soient Σ une base de connaissances, c une clause et $\widehat{\Sigma}_{[]}$ une couverture de simplifications traitables de Σ pour Cs un ensemble de classes traitables pour la compilation modulo des théories.
$\Sigma \models c$ si et seulement si pour toute simplification traitable $p \in \widehat{\Sigma}_{[]}$, telle que $p = \{l_1, l_2, \ldots, l_n\}$:

- *soit $l_1 \wedge l_2 \wedge \ldots \wedge l_n \models c$;*
- *soit $\Sigma_{l_{[1:n]}} \models c$,*

ce qui peut être décidé en temps polynomial par rapport à $|\widehat{\Sigma}_{[]}| + |c|$.

3. Il faut noter que la classe des formules réduites à une conjonction de littéraux est bien traitable pour la compilation modulo une (des) théorie(s).

4. Les couvertures d'impliquants (premiers) ainsi que les couvertures d'impliquants modulo une théorie sont, bine évidemment, des cas particuliers des couvertures traitables.

Preuve (Propriété 12.5)
$\widehat{\Sigma}_{[]}$ est une couverture de simplifications traitables de Σ pour Cs, donc par définition, $(\widehat{\Sigma}_{[]} \wedge \Sigma) \equiv \Sigma$. Ainsi $(\Sigma \vDash c) \Leftrightarrow (\widehat{\Sigma}_{[]} \wedge \Sigma \vDash c) \Leftrightarrow (\forall\, p \in \widehat{\Sigma}_{[]},\ p \wedge \Sigma \vDash c) \Leftrightarrow (\forall\, p \in \widehat{\Sigma}_{[]},\ ((p \vDash c)$ ou $(\Sigma_p \vDash c)))$.
La polynomialité est garantie grâce à l'appartenance de Σ_p à une classe traitable de C de Cs. □

Il faut noter que l'espace requis pour mémoriser une simplification traitable $\{l_1, l_2, \ldots, l_n\}$ est toujours inférieur ou égal à celui nécessaire pour sauvegarder la formule traitable correspondante :

$$\Phi = \{l_1, l_2, \ldots, l_n\} \wedge \Sigma_{l_{[1:n]}} \quad .$$

Cependant, ce gain spatial entraîne un surcoût temporel lors de la phase d'interrogation. Le prix à payer est de $\mathcal{O}(|\Sigma \wedge p|)$ pour chaque simplification traitable $p = \{l_1, l_2, \ldots, l_n\}$, afin de pouvoir retrouver la formule traitable $\Sigma_{l_{[1:n]}}$. Néanmoins, cet accroissement de complexité ne remet pas en question la polynomialité de l'algorithme d'interrogation.

Remarque 12.2
Bien qu'identifier à quelle classe traitable appartient $\Sigma_{l_{[1:n]}}$ est une opération polynomiale, en pratique cette opération n'est pas effectuée lors de la phase d'interrogation. En effet, une fois identifiée lors de la phase de compilation, il est possible d'indexer chaque simplification traitable sauvegardée par une étiquette correspondant à la classe traitable à laquelle la formule appartient. Ce procédé permet un gain substantiel de temps lors des interrogations.

Exemple 12.2 (Exemple de couvertures de simplifications traitables)
Soit $\Sigma = \{(\neg a \vee \neg b \vee \neg c \vee d), (a \vee b \vee d)\}$ et soit Cs l'ensemble de classes traitables contenant la classe des ensembles de clauses de Horn et la classe des ensembles de clauses binaires.
L'ensemble $\widehat{\Sigma}_{[]} = \{\{a\}, \{\neg a\}\}$ est une couverture par simplifications traitables de Σ pour Cs.
En effet, $\Sigma_a = \{(\neg b \vee \neg c \vee d)\}$ est un ensemble de clauses de Horn et $\Sigma_{\neg a} = \{(b \vee d)\}$ est un ensemble de clauses binaires.
D'autre part, si Cs est réduit à une seule classe, celle des formules Horn-renommables, alors l'ensemble vide est une couverture par simplifications traitables de Σ pour Cs puisque Σ est une formule Horn-renommable.

Il est à noter que les compilations basées sur des couvertures de simplifications traitables peuvent conduire à un gain exponentiel d'espace mémoire par rapport aux compilations à base de couvertures d'impliquants premiers : un cas extrême, illustré dans l'exemple ci-dessus, consiste à compiler des bases de connaissances traitables pour lesquelles chaque couverture d'impliquants premiers est de taille exponentielle par rapport à la taille de la base de connaissances.

Nous proposons dans le paragraphe 12.2.4.1 une technique de compilation qui comme pour DPPI ou DPIC utilise une procédure de Davis & Putnam pour collecter les différentes simplifications traitables.

12.2.3 Les couvertures d'hyper-impliquants

Le second cas particulier de couverture traitables que nous avons étudié concerne l'exploitation de la classe traitable des formules Horn-renommables. Cette classe est intéressante à au moins deux égards. Premièrement, elle inclut les ensembles de clauses de Horn, reverse-Horn et binaires et deuxièmement elle est caractérisable par l'existence d'une interprétation spécifique. En effet, dans [Lewis 1978], il est énoncé qu'une formule CNF Σ est Horn-renommable si et seulement s'il existe une interprétation p de Σ telle que

pour chaque clause c de Σ, au plus un littéral de c n'appartient pas à p.

Nous proposons d'utiliser cette caractérisation afin de dériver de Σ des ensembles de clauses Horn-renommables que nous appelons *hyper-impliquants*. Plus formellement, soient Σ un ensemble de clauses (quelconques) et m un modèle de Σ. Pour chaque clause c de Σ, nous notons c^m la clause constituée des littéraux communs à m et c. Par ailleurs, nous notons $P(\Sigma, m)$ l'ensemble de clauses de Σ tel que pour chaque clause c de $P(\Sigma, m)$, c et c^m sont identiques. Nous notons $N(\Sigma, m)$ l'ensemble de clauses complémentaire à $P(\Sigma, m)$ pour Σ, c'est-à-dire l'ensemble de clauses tel que pour chaque clause c de $N(\Sigma, m)$, $c^m \subset c$. Enfin, pour chaque clause c de $N(\Sigma, m)$, nous notons l_c^m un littéral de c tel que $\sim l_c^m$ appartient à m[5]. Étant donné ces conventions, un hyper-impliquant pour un modèle m se définit de la façon suivante :

Définition 12.10 (Couvertures d'hyper-impliquants)

– *Un* **hyper-impliquant** *de Σ pour un modèle m de Σ est une formule notée Σ^m telle que :*

$$\Sigma^m = (\bigwedge_{c \in P(\Sigma,m)} c) \wedge (\bigwedge_{c \in N(\Sigma,m)} (c^m \vee l_c^m))$$

– *Une* **couverture d'hyper-impliquants** *de Σ, notée $\widehat{\Sigma}^m$ est un ensemble d'hyper-impliquants (vus comme leur disjonction) tel que $\Sigma \equiv \widehat{\Sigma}^m$.*

– *Une* **compilation modulo une couverture d'hyper-impliquants** *est le triplet $\langle \Sigma, p_1, Q_{\mathcal{H}} \rangle$ où $p_1(\Sigma) = \widehat{\Sigma}^m$ et $Q_{\mathcal{H}}$ est l'algorithme d'interrogation des formules Horn-renommables.*

Il faut noter que la taille d'un hyper-impliquant de Σ est toujours strictement inférieure à celle de Σ, excepté lorsque Σ est Horn-renommable (dans ce cas Σ et Σ^m peuvent être identiques). Par ailleurs, tout hyper-impliquant est une formule Horn-renommable satisfaisable.

Propriété 12.6

Soient Σ un ensemble de clauses et m un modèle de Σ. L'hyper-impliquant Σ^m de Σ pour le modèle m est une formule Horn-renommable satisfaisable telle que $|\Sigma| \geqslant |\Sigma^m|$.

Preuve (Propriété 12.6)

Les preuves des 3 propriétés : Horn-renommable, satisfaisable et $|\Sigma| \geqslant |\Sigma^m|$ se contruisent très simplement à partir de la construction de Σ^m.

1. Par construction de Σ^m, il est trivial de constater que pour chaque clause c de Σ^m issue de $P(\Sigma, m)$, tous les littéraux de c appartiennent à m. De même pour chaque clause c de Σ^m issue de $N(\Sigma, m)$ seul le littéral l_c^m n'appartient pas à m. Par conséquent, il existe une interprétation (m) qui contredit au plus un littéral par clause de Σ^m, ce qui nous assure que Σ^m est Horn-renommable [Lewis 1978].
2. Toujours par construction, il est clair que chaque clause c de Σ^m est incluse (au sens large) dans une clause de Σ (si c est issue de $P(\Sigma, m)$, alors c appartient également à Σ, si c provient de $N(\Sigma, m)$, alors $c \subseteq c'$ avec $c' \in \Sigma$). Ainsi, m étant un modèle de Σ, m est également un modèle de Σ^m. Par conséquent, Σ^m est satisfaisable.
3. $\forall\ c \in \Sigma^m,\ c \subseteq c'$ avec $c' \in \Sigma$, ce qui signifie que $|c| \leqslant |c'|$. De plus par construction de Σ^m, les clauses de Σ et Σ^m sont en bijection. Par conséquent $|\Sigma| \geqslant |\Sigma^m|$.

□

Remarque 12.3

Il est à noter qu'une étape de sous-sommation peut permettre de diminuer le nombre de clauses de Σ^m. En effet, le choix du littéral l_c^m peut amener à construire des clauses déja présentes dans Σ^m ou sous-

5. Dans le cas général, de toute évidence il peut exister plusieurs candidats l_c^m.

sommées par d'autres clauses de Σ^m ou encore sous-sommant des clauses de Σ^m. Par exemple soient $\Sigma = \{(a \vee b \vee \neg c \vee \neg d), (a \vee b \vee \neg e \vee \neg d), (b \vee \neg e \vee \neg d \vee \neg f)\}$ et $m = \{a, b, c, d, e, f\}$ un modèle de Σ. Si l'on choisit pour chaque clause de prendre comme littéral l_c^m le littéral $\neg d$, nous obtenons $\Sigma^m = \{(a \vee b \vee \neg d), (a \vee b \vee \neg d), (b \vee \neg d)\}$ qui se réduit après sous-sommation à $\{(b \vee \neg d)\}$.

La proposition suivante nous assure qu'une compilation modulo une couverture d'hyper-impliquants préserve l'équivalence.

Propriété 12.7
Soient Σ une base de connaissances, c une clause et $\widehat{\Sigma}^m$ une couverture d'hyper-impliquants de Σ.
$\Sigma \vDash c$ si et seulement si pour toute formule Σ^m de $\widehat{\Sigma}^m$, $\Sigma^m \vDash c$; ce qui est calculé en temps linéaire par rapport à $|\widehat{\Sigma}^m| + |c|$.

Preuve (Propriété 12.7)
Nous avons par définition d'une couverture d'hyper-impliquants : $\Sigma \equiv \widehat{\Sigma}^m$. Donc $\Sigma \vDash c \Leftrightarrow \widehat{\Sigma}^m \vDash c \Leftrightarrow \forall\, \Sigma^m \in \widehat{\Sigma}^m,\ \Sigma^m \vDash c$.
Plusieurs algorithmes linéaires ont été proposés pour tester la satisfaisabilité d'une formule Horn-renommable [Lewis 1978, Aspvall 1980, Chandru *et al.* 1990, Hébrard 1994]. Comme les instances Horn-renommables sont stables par adjonctions de clauses unitaires, tester la satisfaisabilité de $\Sigma^m \wedge \neg c$ est une opération réalisable en temps linéaire par rapport à $|\Sigma^m| + |c|$. Par conséquent $\forall\, \Sigma^m \in \widehat{\Sigma}^m,\ \Sigma^m \vDash c$ se calcule en temps linéaire par rapport à $|\widehat{\Sigma}^m| + |c|$. □

Le principal obstacle des couvertures d'hyper-impliquants demeure dans l'espace mémoire requis pour sauvegarder, l'ensemble des formules Horn-renommables équivalent à Σ. Nous proposons une technique de mémorisation de la couverture en fonction d'un ordre arbitraire sur les clauses de Σ. Supposons que les clauses de $\Sigma = \{c_1, c_2, \ldots, c_n\}$ soient totalement ordonnées ; une couverture d'hyper-impliquants de Σ peut alors se représenter par une ensemble de couples $\langle m, [l_1^m, l_2^m, \ldots, l_n^m]\rangle$ où m est un modèle de Σ et l_i^m $(i \in [1..n])$ est $\bot$ si $c_i \in P(\Sigma, m)$ et est le littéral de $c_i \setminus m$ conservé sinon. Cette manière de procéder permet d'obtenir un gain substantiel d'espace. En contrepartie, lors de l'interrogation, il faudra reconstruire chaque Σ^m à partir de chaque $\langle m, [l_1^m, l_2^m, \ldots, l_n^m]\rangle$ mémorisé. Il faut noter que cette opération est réalisable en temps linéaire et permet de conserver la polynomialité de l'interrogation.

Intuitivement, un hyper-impliquant Σ^m de Σ permet de représenter de manière concise un sous-ensemble d'impliquants premiers de Σ. Plus précisément, les impliquants premiers de Σ obtenus à partir de m sont des impliquants de Σ^m.

Propriété 12.8
Pour chaque modèle m de Σ, soit $PI_m(\Sigma)$ l'ensemble des impliquants premiers de Σ tel que $\forall\ \pi \in PI_m(\Sigma),\ m \vDash \pi$. Si $PI(\Sigma^m)$ représente l'ensemble des impliquants premiers de l'hyper-impliquant Σ^m issu du modèle m de Σ, alors $PI_m(\Sigma) \subseteq PI(\Sigma^m)$.

Preuve (Propriété 12.8)
L'objectif est de montrer que pour chaque clause c de Σ^m et pour chaque impliquant premier π de Σ tel que $m \vDash \pi$, nous avons $\pi \vDash c$.
Soit π un impliquant premier de Σ tel que $m \vDash \pi$. Quelle que soit la clause c de Σ^m :

- soit c est issue de $P(\Sigma, m)$ auquel cas $c \in \Sigma$ et par conséquent $\pi \vDash c$ (π impliquant premier de Σ) ;
- soit c est issue de $N(\Sigma, m)$ auquel cas il existe une clause c' de Σ telle que $c' = (c \vee l_1 \vee l_2 \vee \ldots \vee l_n)$ avec $l_i \notin m,\ i \in [1..n]$. Par ailleurs $(m \vDash \pi) \Leftrightarrow (\pi \subseteq m)$, donc $l_i \notin \pi,\ i \in [1..n]$. De plus, π est impliquant premier de Σ, donc $\pi \vDash c'$.
  $(\pi \vDash c') \Leftrightarrow (\pi \vDash (c \vee l_1 \vee l_2 \vee \ldots \vee l_n)) \Leftrightarrow ((\pi \vDash c)$ ou $(\pi \vDash l_1 \vee l_2 \vee \ldots \vee l_n))$. Nous avons

$l_i \not\in \pi,\ i \in [1..n]$, donc $(\pi \not\models l_1 \vee l_2 \vee \ldots \vee l_n)$, ainsi $(\pi \models c)$.
Par conséquent, quel que soit π de $PI_m(\Sigma)$, $\pi \models \Sigma^m$, ce qui signifie que π est une impliquant de Σ^m. D'autre part, quelle que soit $c \in \Sigma^m$, $c \subseteq c'$ avec c' clause de Σ, si π est premier pour Σ, alors il l'est également pour Σ^m. Ainsi pour tout π de $PI_m(\Sigma)$, π est un impliquant premier de Σ^m. □

La propriété précédente nous assure donc que Σ^m couvre chaque modèle de Σ couvert par un impliquant premier construit à partir de m. D'autre part, nous montrons qu'une couverture d'hyper-impliquants offre une représentation plus écononique (en term d'espace mémoire) qu'une couverture d'impliquants premiers. Hormis le fait que les hyper-impliquants de Σ sont plus petits que Σ, le nombre d'hyper-impliquants d'une couverture d'hyper-impliquants est plus petit que le nombre d'impliquants premiers dans une couverture d'impliquants premiers.

Propriété 12.9
Pour toute couverture d'impliquants premiers de Σ contenant t impliquants premiers, il existe une couverture d'hyper-impliquants de Σ contenant au plus $[\frac{t+1}{2}]$ hyper-impliquants ($[x]$ représentant la partie entière de x).

Preuve (Propriété 12.9)
L'idée est de construire à partir de deux impliquants premiers (π_1 et π_2) de la couverture d'impliquants premiers, un hyper-impliquant Σ^m de Σ couvrant π_1 et π_2.
Pour la couverture d'impliquants premiers de Σ, on regroupe chaque impliquant premier par couple de telle manière qu'un impliquant n'appartienne qu'à un et un seul couple[6]. Pour chaque couple d'impliquants premiers (π_1, π_2), l'ensemble de clauses suivant est formé par les deux étapes ci-dessous :

1. pour chaque clause c de Σ, nous construisons $c_1 = (c \cap \pi_1)$, c'est-à-dire la clause contenant uniquement les littéraux de c satisfaits par π_1 ;
2. pour chaque clause c_1 (formée en 1) qui n'est pas satisfaite par π_2, on ajoute un seul littéral[7] de la clause initiale c satisfait par π_2.

L'ensemble de clauses Σ^m ainsi construit est bien une formule Horn-renommable satisfaisable (la Horn renommabilité et la satisfaisabilité sont assurées par π_1 qui est un modèle de Σ^m tel que toutes les clauses de Σ^m contiennent au plus un littéral insatisfait[8] par π_1). Par ailleurs, Σ^m est construit de telle manière que chaque clause c de Σ^m est incluse dans une clause c' de Σ. Par conséquent, Σ^m est un hyper-impliquant de Σ.
Par construction, il est clair que Σ^m couvre les modèles couverts par π_1 et π_2.
Par ailleurs, s'il y a un impliquant isolé (nombre impair d'impliquants premiers dans la couverture), nous construisons n'importe quel hyper-impliquant le couvrant (par exemple celui obtenu à partir de l'étape 1). Ainsi l'ensemble d'hyper-impliquants construits pour chaque couple constitue bien une couverture d'hyper-impliquants de Σ, dont la taille est de $[\frac{t+1}{2}]$, avec t le nombre d'impliquants premiers contenus dans la couverture d'impliquants premiers. □

Nous illustrons cette propriété sur l'exemple suivant.

Exemple 12.3 (Exemple d'une couverture d'hyper-impliquants)
Soit $\Sigma = \{(a \vee c \vee d), (a \vee \neg b \vee c), (a \vee b \vee \neg c \vee \neg d), (\neg a \vee \neg b \vee \neg c \vee \neg d), (\neg a \vee b \vee \neg c \vee d)\}$.
Une couverture d'hyper-impliquants de Σ se représente par les deux couples :

- $\langle\{a, \neg b, \neg c, d\}, [c, c, b, \neg a, \neg a]\rangle$;
- $\langle\{\neg a, b, c, \neg d\}, [a, \neg b, \neg c, \neg b, d]\rangle$.

6. Si le nombre d'impliquants premiers de la couverture est impair, un impliquant premier est isolé et un hyper-impliquant sera construit spécialement pour couvrir cette impliquant premier.
7. Il en existe toujours au moins un.
8. Le littéral ajouté lors de la deuxième étape.

De cette représentation on extrait les deux hyper-impliquants (formules Horn-renommables) :

– $\{(a \vee d \vee c), (a \vee \neg b \vee c), (a \vee \neg c \vee b), (\neg b \vee \neg c \vee \neg a), (\neg c \vee d \vee \neg a)\}$;

– $\{(c \vee a), (c \vee \neg b), (b \vee \neg d \vee \neg c), (\neg a \vee \neg d \vee \neg b), (\neg a \vee b \vee d)\}$.

La couverture d'hyper-impliquants proposée contient 2 formules alors que la plus petite couverture d'impliquants premiers contient au mimimum 6 impliquants premiers.

Par ailleurs, les expériences réalisées sur les compilations à base de couvertures d'hyper-impliquants montrent, pour la plupart des bases de connaissances testées, une réduction significative de l'espace mémoire nécessaire à la mémorisation des couvertures d'hyper-impliquants comparativement à celui requis par les couvertures d'impliquants premiers (cf. paragraphe 12.2.5 page 196).

12.2.4 Algorithmes

Comme pour la collecte d'impliquants modulo une théorie, nous proposons un algorithme `DPTC` (cf. algorithme 12.1) pour calculer des couvertures traitables (représentées sous forme d'interprétations partielles) basé sur la procédure de Davis & Putnam. Cette procédure reste proche de DPPI proposée dans [Schrag 1996].

Les fonctions `Propagation_Unitaire` et `Heuristique_de_branchement` sont standard à toute procédure de Davis & Putnam, nous les avons largement détaillées dans le chapitre 7. Le rôle principal de notre procédure `DP*` est de trouver les impliquants de Σ en parcourant un arbre de recherche complet. Les procédures `PRUNING` et `PROCESS_IMPLICANT` dépendent de l'approche considérée, nous les détaillons pour les couvertures de simplifications traitables et pour celles d'hyper-impliquants dans les deux prochains paragraphes. Les éléments de la couverture calculée (simplifications traitables ou hyper-impliquants) sont extraits des impliquants par l'intermédiaire de la procédure `PROCESS_IMPLICANT` et chaque élément ainsi collecté est mémorisé dans la variable globale `PC`. La fonction `PRUNING` permet d'élaguer l'arbre de recherche en évitant de parcourir des sous-arbres qui ne pourraient amener qu'à des impliquants déjà couverts par la couverture partielle calculée.

Algorithme 12.4 DPTC

```
1.  Procedure DPTC
2.  Input  : une formule CNF Σ et un ensemble Cs de classes traitables ;
3.  Output : une représentation des connaissances équivalentes à Σ ;
4.  Begin
5.    PC=∅ ;                                          %% PC : couverture partielle
6.    DP*(Σ, Cs, ∅) ;              %% DP* est un DP «classique» modifié pour récolter
        %% des formules traitables représentées sous la forme d'interprétations partielles
7.    return ⟨Σ,PC⟩ ;
8.  End

1.  Procedure DP*
2.  Input  : une formule CNF Σ, un ensemble de classes traitables Cs
             et une interprétation partielle p ;
3.  Output : modifie la variable globale PC (la couverture partielle courante) ;
4.  Begin
5.    if (not(PRUNING(Σ, p)))
6.    then
7.       Σ=Propagation_Unitaire(Σ, p) ;   %% La procédure de propagation unitaire a été
                  %% modifiée de manière à mémoriser dans p chaque  propagation réalisée
8.       if (Σ contient la clause vide) then return ;
9.       elif (p ⊨ Ψ)
10.        PROCESS_IMPLICANT(Σ, Cs, p) ;
11.        return ;
12.      else
13.        l=Heuristique_de_branchement(Σ) ;
14.        DP*((Σ ∧ (l)), p ∪ {l});
15.        DP*((Σ ∧ (¬l)), p ∪ {¬l});
16.      fi
17.   fi
18. End
```

12.2.4.1 Le cas des simplifications traitables

Nous détaillons dans l'algorithme 12.5 page suivante les procédures spécifiques au calcul d'une couverture de simplifications traitables.

À chaque fois qu'un impliquant p de Σ est trouvé, un impliquant premier Prime de Σ est extrait de p. Prime est calculée par l'intermédiaire de la fonction ONE_PRIME où chaque littéral de p est considéré successivement afin de déterminer s'il est nécessaire à la satisfaction de Σ. Diverses procédures plus ou moins sophistiquées d'extraction d'impliquants premiers peuvent être trouvées dans la littérature [Castell & Cayrol 1996b, Schrag 1996]. Une fois extrait un impliquant premier de Σ issu de p, cet impliquant premier (assimilé à une interprétation partielle) est passé à la procédure DERIVE_{ST} qui va chercher à simplifier (supprimer des littéraux) cette interprétation partielle afin d'obtenir une simplification traitable, c'est-à-dire une interprétation partielle ST telle que Σ_{ST} soit une formule appartenant à une des classes C de Cs. Comme pour ONE_PRIME, chaque littéral l de Prime est considéré successivement et l'on vérifie par l'intermédiaire des algorithmes de reconnaissance de chaque classe C traitable de Cs si $\Sigma_{\text{Prime}\setminus\{l\}}$ appartient à au moins une classe de C de Cs. Si $\Sigma_{\text{Prime}\setminus\{l\}}$ est prouvée appartenir à une classe C de Cs alors le littéral l est retiré de Prime et le processus est réitéré avec la nouvelle interprétation partielle $\text{Prime}\setminus\{l\}$.

Lorsque chaque littéral de Prime a été considéré, l'interprétation partielle résultante est retournée à la procédure PROCESS_IMPLICANT$_{ST}$. Seules les simplifications traitables maximales pour $\models$ doivent être sauvegardées dans PC. Cette condition est assurée par les deux caractéristiques suivantes. D'une part, l'ensemble PC de simplifications traitables est utilisé durant l'exploration de l'arbre de recherche de DP afin d'élaguer cet arbre. Cet élagage, effectué par l'intermédiaire de la fonction PRUNING$_{ST}$, a pour conséquence d'éviter l'exploration de sous-arbres qui ne pourraient qu'aboutir qu'à des interprétations déjà couvertes par PC. D'autre part, à chaque nouvel ajout d'une simplification traitable dans PC, les interprétations couvertes par ce nouvel élément sont retirées de PC (ligne 6 de PROCESS_IMPLICANT$_{ST}$).

Algorithme 12.5 DPTC-SIMPLIFICATIONS-TRAITABLES

```
Function PRUNING_ST : boolean
Input  : Un ensemble de clauses Σ et p l'interprétation partielle courante ;
Output : true s'il existe une simplification traitable de PC couvrant p,
         false sinon ;
1.  Begin
2.    foreach interprétation partielle π ∈ PC do
3.      if (p ⊨ π) then return true ;
4.    done
5.    return false ;
6.  End

Procedure PROCESS_IMPLICANT_ST
Input  : Un ensemble de clauses Σ, un ensemble de classes traitables Cs
         et un impliquant p de Σ ;
Output : Modifie la variable globale PC représentant la couverture courante ;
1.  Begin
2.    Prime = ONE_PRIME(Σ, p) ;
3.    ST = DERIVE_ST(Σ, Cs,Prime) ;  %% Extraction d'une simplification traitable (ST) à
                                     %% partir d'un impliquant premier (Prime) issu de p.
                %% Chaque ST est sauvegardée sous la forme d'une interprétation partielle.
4.    foreach interprétation π ∈ PC
5.    do                             %% Assure l'obtention d'une couverture irredondante
6.      if (π ⊨ST) then  PC = (PC\{π}) ;
7.    done
8.    PC = (PC ∪ {ST}) ;
9.  End

Function DERIVE_ST : une simplification traitable
Input  : Un ensemble de clauses Σ, un ensemble de classes traitables Cs
         et un impliquant premier (Prime) de Σ
Output : Une simplification traitable de Σ pour Cs ;
1.  Begin
2.    foreach littéral l de Prime
3.    do
4.      foreach classe traitable C(=⟨T_C, Q_C⟩) de Cs
5.      do
6.        if (T_C(Σ_Prime\{l})) then return DERIVE_ST(Σ, Cs,Prime\{l}) ;
7.      done
8.    done
9.    return Prime ;
10. End
```

Il est clair, au vu de ces procédures, que l'ordre de considération des littéraux et les procédures de reconnaissance d'instances traitables utilisées dans la procédure $\texttt{DERIVE}_{ST}$ ont une grande influence sur la couverture générée par notre technique et par conséquent sur l'efficacité globale d'une telle approche. Par ailleurs, il est à noter que pour des raisons d'efficacité chaque simplification traitable ST retournée par $\texttt{DERIVE}_{ST}$ est indexée par la classe traitable C à laquelle appartient la formule $\Sigma_{\texttt{ST}}$ (cf. remarque 12.2 et algorithme 12.6).

L'algorithme d'interrogation par une clause de c lié à la compilation modulo une couverture de simplifications traitables est détaillé ci-dessous. Il consiste pour chaque simplification traitable π, à s'assurer que soit π implique directement c soit Σ_π implique c.

Algorithme 12.6 DPTC-ST-QUERY

```
1.  Function DPTC_ST_QUERY : boolean
2.  Input  : Σ un ensemble de clauses, une clause c à interroger et PC une
             couverture de simplifications traitables de Σ pour un ensemble de
             classes traitables Cs ;
3.  Output : vrai si Σ ⊨ c, faux sinon ;
4.  Begin
5.    foreach π ∈ PC t.q. π est indexée par la classe traitable C de Cs
6.    do
7.       if (c ∩ π==∅)                    %% Si c n'est pas directement impliquée par π
8.       then
9.         if (not(T_C(Σ_π, c)))
10.        then
11.           return false ;
12.        fi
13.      fi
14.   done
15.   return true ;
16. End
```

Nous présentons dans la section 12.2.5 une étude comparative des trois approches de compilation discutées précédemment :

- couverture d'impliquants premiers ;
- couverture de simplifications traitables ;
- couverture d'hyper-impliquants.

Ces trois approches sont également comparées par rapport à l'interrogation directe de la base de clauses.

12.2.4.2 Le cas des hyper-impliquants

Le pseudo-code page suivante décrit les procédure spécifiques à la compilation modulo une couverture d'hyper-impliquants. Il est à noter que, pour le calcul d'hyper-impliquants, Cs est vide. En effet, seul des formules Horn-renommables sont sauvegardées par l'intermédiraire d'un couple (interprétation partielle,

ensemble de littéraux) comme nous l'avons explicité précédemment (cf. paragraphe 12.2.3).

Algorithme 12.7 DPTC-HYPER-IMPLIQUANTS

```
Function PRUNING_HI : boolean
Input  : Un ensemble de clauses Σ et p l'interprétation partielle courante ;
Output : true s'il existe un hyper-impliquant de PC couvrant Σ^p,
         false sinon ;
1.  Begin
2.    foreach interprétation partielle π ∈ PC
3.    do
4.      if (Σ^p ⊨ Σ^π) then return true ;
5.    done
6.    return false ;
7.  End

Procedure PROCESS_IMPLICANT_HI
Input  : Un ensemble de clauses Σ et un impliquant p de Σ ;
Output : Modifie la variable globale PC représentant la couverture courante ;
1.  Begin
2.    Model = DERIVE_HI(Σ, p) ;
3.    foreach interprétation π ∈ PC
4.    do
5.      if (Σ^Model ⊨ Σ^π) then  PC = (PC\{π}) ;
6.    done
7.    PC = (PC ∪ {ST}) ;
8.  End

Function DERIVE_HI : une simplification traitable
Input  : Un ensemble de clauses Σ et un impliquant p de Σ
Output : Un hyper-impliquant de Σ pour p ;
1.  Begin
2.    Model = p ;
3.    foreach variable v de Σ telle que {v} ∩ p = {¬v} ∩ p = ∅
4.    do
5.      if (nombre d'occurrences de v dans Σ est supérieur à celui de ¬v)
6.      then
7.         Model = (Model ∪ {v}) ;
8.      else
9.         Model = (Model ∪ {¬v}) ;
10.     fi
11.   done
12.   return Model ;
13. End
```

À chaque fois qu'un impliquant p de Σ est trouvé par DP, cet impliquant est étendu vers un modèle `Model` (une interprétation complète) de Σ tel que $\texttt{Model} \models p$. Les variables n'apparaissant pas dans p sont ajoutées sous la forme d'un littéral, le signe de ce littéral est déterminé en fonction du déséquilibre de la variable. Lorsque $\Sigma^{\texttt{Model}}$ est calculé pour la première fois, la liste de littéraux $[l_1^m, l_2^m, \ldots, l_n^m]$ est greffée à `Model`. La procédure `DERIVE_HI` est à la fois plus simple et beaucoup plus efficace que la procédure `DERIVE_ST`. De plus, la méthode n'oblige pas le calcul d'impliquants premiers, par conséquent elle fait

l'économie de l'appel à ONE_PRIME. Comme pour les couverures de simplifications traitables, la couverture partielle courante permet d'élaguer l'arbre de recherche de DP, par l'intermédiaire de la procédure PRUNING_HI. Il reste que cette procédure, à l'inverse de DERIVE_HI, est beaucoup moins efficace que PRUNING_ST. En effet, pour chaque modèle sauvegardé, il faut calculer l'hyper-impliquant correspondant et tester l'implication de deux formules Horn-renommables. De plus, si l'on désire que la couverture calculée soit maximale pour $\models$, il ne faut pas sauvegarder des hyper-impliquants qui seraient couverts par d'autres hyper-impliquants. Dans ce cas le test d'implication entre deux formules Horn-renommables ($\Sigma^{\texttt{Model}} \models \Sigma^{\pi}$) fait en ligne 5 de la procédure DERIVE_HI est indispensable. Cependant, étant donné que les formules $\Sigma^{\texttt{Model}}$ et Σ^{π} sont issues toutes deux de Σ, ce test peut être calculée de manière relativement efficace (comparativement à un test d'implication entre deux formules Horn-renommables quelconques). Il reste que, même si ces tests d'implications peuvent être calculés de manière relativement efficace, ils risquent fortement de nuire aux performances globales de l'approche. Par conséquent, ces tests coûtant la plupart du temps plus chers qu'ils ne rapportent, sont à proscrire en pratique.

Similairement au calcul d'une simplification traitable où l'ordre d'examen des littéraux dans la procédure DERIVE_ST et les classes traitables contenues dans Cs peuvent grandement influencer la simplification calculée, les couvertures d'hyper-impliquants admettent également deux points critiques. En effet, le choix des littéraux pour étendre l'impliquant p de Σ vers un modèle dans DERIVE_HI, ainsi que le choix des littéraux l_c^m peuvent avoir un impact non négligeable sur l'hyper-impliquant généré.

L'algorithme d'interrogation pour les compilations modulo une couverture d'hyper-impliquants est le plus simple et le plus naturel qui soit.

Algorithme 12.8 DPTC-HI-QUERY

```
1.  Function DPTC_HI_QUERY : boolean
2.  Input  : Σ un ensemble de clauses, une clause c à interroger et PC une
             couverture d'hyper-impliquants de Σ ;
3.  Output : vrai si Σ ⊨ c, faux sinon ;
4.  Begin
5.    foreach π ∈ PC t.q. π
6.    do
7.       if (Σ^π ⊭ c) then return false ;
8.    done
9.    return true ;
10. End
```

12.2.5 Résultats expérimentaux

Afin d'évaluer expérimentalement les approches de compilation logique à base de couvertures traitables, nous avons comparé nos approches à base de simplifications traitables et d'hyper-impliquants avec celles utilisant une couverture d'impliquants premiers. Plus particulièrement, la technique de compilation à base de couvertures d'impliquants premiers retenue est celle proposée par Robert C. Schrag dans [Schrag 1996].

Nos expériences ont porté sur de nombreuses bases de connaissances incluant des problèmes structurés issus de [Forbus & De Kleer 1993] et d'instances *k***SAT** aléatoires, pour un rapport nombre de clauses sur nombre de variables variant entre 3.2 et 4.4.

problèmes	#var	#cla	Impliquants Premiers			Simplifications Traitables			Hyper-Impliquants		
			α	β	$\lvert\widehat{\Sigma}\rvert$	α	β	$\lvert\widehat{\Sigma}_{[]}\rvert$	α	β	$\lvert\widehat{\Sigma}^m\rvert$
adder	21	50	5.08	—	5376	2.09	—	1044	0.20	78.75	305
history-ex	21	17	2.00	—	172	0.75	1000.00	234	0.14	11.66	77
two-pipes	15	54	0.66	125.00	255	0.60	125.00	234	0.12	20.00	192
three-pipes	21	82	0.47	45.45	441	0.42	<1	373	0.27	17.50	284
four-pipes	27	110	0.36	23.80	675	0.25	18.51	528	0.20	29.16	380
regulator	21	106	0.51	26.31	861	0.29	20.83	530	0.17	29.99	422
selenoid	11	19	2.16	—	297	1.40	—	115	0.20	17.49	94
valve	13	50	0.85	<1	312	0.66	<1	297	0.16	21	246

TAB. 12.1 – *Résultats des différentes compilations sur des instances structurées*

Chaque base de connaissances a été compilée suivant les trois techniques et interrogée par 500 questions (clauses de longueur 3) générées aléatoirement. Par ailleurs, afin de s'assurer de l'utilité de compiler ces bases de connaissances, nous avons également interrogé directement chaque base de connaissances par les 500 questions générées. Nous rappelons que l'interrogation directe consiste à tester la consistance de la formule $(\Sigma \wedge \neg c)$ où Σ est l'ensemble de clauses à interroger et c une question exprimée sous forme clausale. L'algorithme complet utilisé pour cette interrogation directe est C-SAT [Dubois *et al.* 1996], l'une des meilleures implantations de DP (cf. paragraphe 7.1.4.1).

Pour chaque problème testé, nous avons calculé les rapports suivants :

- $\alpha = \frac{Q_C}{Q_D}$ où Q_C (respectivement Q_D) correspond au temps (moyen) nécessaire aux approches compilées (respectivement à l'interrogation directe) pour répondre aux 500 requêtes ;
- $\beta = \frac{C}{Q_D - Q_C}$ où C correspond au temps de compilations.

α nous indique la rentabilité de l'interrogation via une approche compilée et β nous indique le nombre de questions nécessaires pour amortir le coût de la compilation. Si $\alpha > 1$ alors l'approche directe sera toujours plus efficace que l'approche compilée donc la compilation ne sera jamais amortie.

La table 12.1 rapporte les résultats obtenus par chacune des méthodes de compilations testées pour chaque problème issu de [Forbus & De Kleer 1993]. Nous avons spécifié pour chacune des méthodes les ratios α et β ainsi que la taille de la compilation. La taille de la compilation est calculée en nombre de littéraux de la couverture. La taille des couvertures traitables est calculée de manière à prendre en compte la taille de la représentation sous forme de disjonctions d'interprétations partielles et la taille de l'instance Σ nécessaire pour retrouver les formules traitables à partir des interprétations partielles sauvegardées. Pour

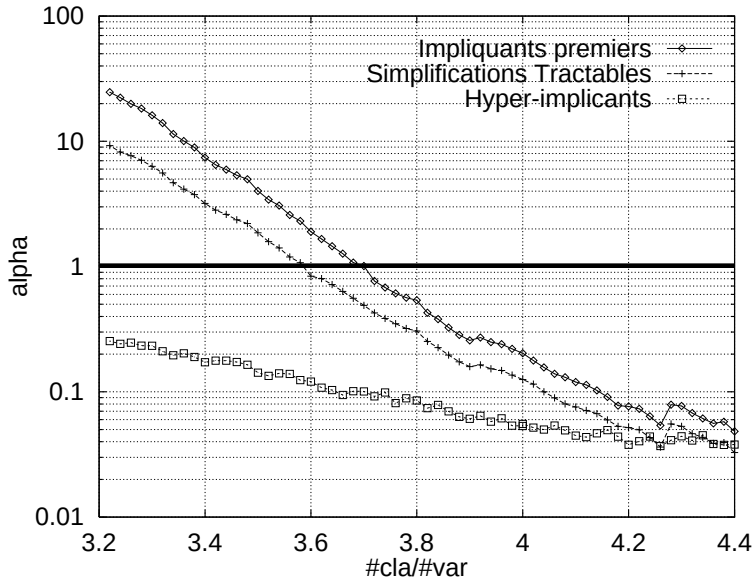

FIG. 12.5 – *Valeur de α pour des instances 3**SAT** aléatoires en fonction de C/V*

l'approche à base de simplifications traitables, Cs comprend la classe des ensembles de clauses de Horn, reverse-Horn et binaires. Pour l'approche à base d'hyper-impliquants la simplification de la couverture (lignes de 3 à 6 de PROCESS_IMPLICANT$_{HI}$) n'a pas été implantée.

Le second type de problèmes testés sont des instances 3**SAT** aléatoires de 50 variables. Pour ces instances, nous avons fait varier par pas de 0.01 le rapport nombre de clauses sur nombre de variables entre 3.2 et 4.4, ceci permet de couvrir au mieux toutes les zones (sous-contraintes, critiques, sur-contraintes) des instances 3**SAT** aléatoires satisfaisables. Pour chaque taille de problèmes, 50 instances ont été générées. Les courbes 12.5 et 12.6 donnent repectivement la valeur moyenne de α (respectivement la taille moyenne des compilations) pour chacune des tailles d'instances testées. Sur la courbe correspondant au ratio α (cf. FIG. 12.5), nous avons partagé l'espace en deux régions par l'intermédiaire de la droite $\alpha = 1$. Ces régions correspondent pour la zone située au dessous (respectivement au dessus) de la droite à la région où la compilation est utile (respectivement inutile), c'est-à-dire la région où la compilation permet (repectivement ne permet pas) d'obtenir un gain par rapport à l'interrogation directe .

Au vu des expériences réalisées, les couvertures traitables fournissent de meilleurs résultats que les couvertures d'impliquants premiers à la fois pour les instances structurées et aléatoires. Un gain de temps significatif lors de la phase d'interrogation et un gain d'espace mémoire pour mémoriser la couverture ont été obtenus. De plus, les couvertures traitables sont utiles pour un grand nombre d'instances (où elles sont plus efficaces que l'interrogation directe). Enfin les couvertures traitables permettent de compiler des bases de connaissances admettant une couverture d'impliquants premiers (zone sous-contraintes des instances $kSAT$ aléatoires) si grande qu'il est impossible de la calculer et de la mémoriser.

12.2.6 Conclusion

Les résultats théoriques et expérimentaux obtenus montrent que les approches de compilations à base de couvertures traitables sont plus que prometteuses et nous encouragent à étendre ces travaux dans différentes directions.

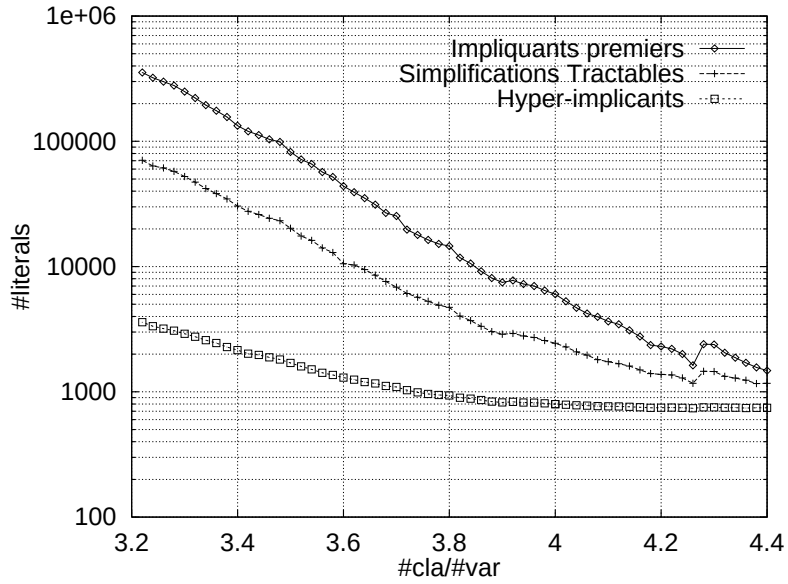

FIG. 12.6 – *Taille des compilations pour des instances* 3**SAT** *aléatoires en fonction de C/V*

Une première voie quant à l'extension de ces travaux est de mettre au point une technique permettant de déterminer quelles sont les classes traitables à utiliser pour une base de connaissances donnée. D'un point de vue expérimental, une évaluation intensive de l'approche à base de simplifications traitables utilisant un ensemble de classes traitables plus important doit être réalisée.

Une autre perspective intéressante concerne l'extension de l'approche à base d'hyper-impliquants à d'autres classes traitables. Nous pensons particulièrement à la classe traitable q-Horn de **SAT** [Boros *et al.* 1994].

Enfin, toutes les approches proposées présentent la caractéristique d'être « *anytime* », dans le sens où lorsque la compilation est stoppée avant sa terminaison naturelle, la couverture partielle obtenue peut jouer le rôle de minorant (cf. paragraphes 11.2.2.1 et 11.2.2.2). Les approximations ainsi construites pourront être plus fines comparativement aux approches classiques d'approximation par une formule Horn [Selman & Kautz 1991, Del Val 1996], puisque la couverture partielle calculée contient plusieurs formules traitables issues de différentes classes traitables.

Chapitre 13

Conclusions – Perspectives

Notre contribution à l'étude des techniques de compilation logique de bases de connaissances en logique propositionnelle porte sur la mise au point de nouvelles approches à base d'un raisonnement modulo des théories. Plus précisément, nous avons proposé différentes techniques de compilation logique qui fournissent une représentation de la base de connaissances sous la forme d'une disjonction de formules traitables (les théories). La nouvelle représentation de la connaissance obtenue à partir d'une compilation modulo des théories est garantie « interrogeable » en temps polynomial, c'est-à-dire que toute déduction via cette représentation peut s'effectuer en temps polynomial par rapport aux tailles cumulées de la représentation et de la clause à déduire.

Les compilations modulo des théories reposent sur de nouveaux concepts, dont certains constituent une généralisation de concepts largement utilisés en logique propositionnelle. En effet, les impliquants d'une formule peuvent être assimilés à des impliquants modulo la théorie vide.

L'idée centrale des compilations modulo des théories est de construire une couverture traitable, c'est-à-dire un ensemble de formules (vues comme leur disjonction) $\mathcal{T} = \{\Sigma\pi_1, \Sigma\pi_2, \ldots, \Sigma\pi_n\}$ tel que si Σ est la base de connaissances à compiler, alors $\Sigma \equiv \mathcal{T}$. De plus chaque formule Σ_{π_i} ($i \in [1..n]$) appartient à une classe traitable pour la compilation logique (existence d'algorithmes polynomiaux en temps pour la vérification d'appartenance à la classe et pour la satisfaisabilité d'une formule de la classe, et stabilité de la classe par adjonctions de clauses unitaires). La représentation des connaissances ainsi construite est donc bien interrogeable en temps polynomial : pour toute clause c, $\Sigma \models c$ si et seulement si $\forall i \in [1..n],\ \Sigma_{\pi_i} \models c$, opération assurée polynomiale par l'appartenance de Σ_{π_i} à une classe traitable.

Hormis le cas particulier des couvertures d'impliquants premiers, nous avons proposé trois nouvelles approches de compilation à base de couvertures traitables :

1. les couvertures modulo une théorie ;
2. les couvertures de simplifications traitables ;
3. les couvertures d'hyper-impliquants.

Pour les couvertures modulo une théorie, l'idée est de scinder la base de connaissances Σ en deux parties une facile Φ et une difficile Ψ de telle manière que $\Sigma \equiv (\Phi \wedge \Psi)$ et que la partie facile soit traitable. Une fois cette opération effectuée une couverture d'impliquants de Ψ modulo la théorie Φ est calculée. Nous avons montré qu'une telle couverture pouvait être utilisée pour interroger polynomialement Σ par n'importe quelle clause.

Les couvertures de simplifications traitables consistent à collecter une série d'interprétations partielles π_i appelée simplification traitable telle que Σ_{π_i} soit une formule traitable.

Les couvertures d'hyper-impliquants sont basées sur la caractérisation des formules Horn-renommables (existence d'un modèle contredisant au plus un littéral par clause). L'idée est donc ici de collecter des modèles π_i de la base de connaissances et de construire une formule Σ^{π_i} Horn-renommable appelée hyper-impliquant.

Pour ces trois types de couvertures traitables, nous avons proposé des algorithmes de calcul simples et efficaces basés sur la procédure de Davis & Putnam. Les résultats expérimentaux obtenus par ces approches montrent, pour une large classes d'instances, que les compilations modulo des théories sont viables en pratique puisque plus efficaces que l'interrogation directe de bases de connaissances et que l'espace mémoire requis pour sauvegarder ces compilations est très nettement inférieur à celui des compilations d'impliquants premiers, ce qui a pour conséquence directe de diminuer sensiblement le temps d'interrogation. Par ailleurs, il a été montré que pour une couverture d'impliquants premiers donnée contenant t impliquants premiers, il existe toujours une couverture d'hyper-impliquants contenant, au pire, $\frac{t+1}{2}$ hyper-impliquants. D'autre part, quelle que soit l'approche de couvertures traitables utilisée, les méthodes proposées sont « *anytime* » dans le sens où si l'une d'elles est stoppée avant sa terminaison naturelle la couverture traitable partielle calculée est un minorant de la base de connaissances : si une clause n'a pu être impliquée par la couverture traitable partielle alors a fortiori elle ne le sera pas non plus par une couverture traitable complète et donc par la base de connaissances.

Les résultats théoriques et expérimentaux obtenus quant aux approches à base de couverture traitables sont très prometteurs et nous encouragent à étendre ces travaux dans différentes directions.

Les algorithmes proposés se basant sur la procédure de Davis & Putnam, il serait intéressant de déterminer des stratégies de branchements pour cette procédure permettant de faciliter l'obtention de formules traitables, notamment en ce qui concerne les couvertures de simplifications traitables. D'autre part, pour ce type de couverture, l'ensemble de classes traitables considéré est relativement de petite taille, il serait intéressant d'étendre cet ensemble à un nombre de classes plus important, ce qui devrait permettre sans nul doute de réduire encore les tailles des compilations. Enfin les couvertures d'hyper-impliquants reposant sur la caractérisation des formules Horn-renommables, une perspective intéressante concerne la mise au point de techniques spécifiques basées sur la caractérisation d'autres classes traitables.

Conclusion générale

Dans ce mémoire de thèse, nous avons présenté une étude réalisée sur les problème de satisfaisabilité et de déduction en logique propositionnelle.

Pour le problème de la satisfaisabilité notre contribution porte essentiellement sur les aspects algorithmiques. Dans ce domaine, plusieurs approches ont été proposées : incomplètes, complètes et mixtes.

- En ce qui concerne les approches incomplètes, nous avons proposé un nouvel algorithme de recherche locale, TSAT, basé sur le concept du tabou. Les résultats obtenus quant à l'optimisation expérimentale de cette méthode ont permis de mettre en avant la linéarité de la courbe de la meilleure taille de la liste tabou en fonction du nombre de variables pour des instances *k***SAT** aléatoires. D'autre part, la technique proposée fournit d'excellents résultats rivalisant avec les meilleures approches actuelles de recherche locale. Diverses extensions ont déja été réalisées (ou sont en cours de réalisation) sur ce travail, comme par exemple la mise au point d'une technique à base de tabou dynamique. L'utilisation du tabou dans les méthodes de recherche locale permet de supprimer le caractère aléatoire des meilleures techniques de réparations locales actuelles. Nous pensons qu'à l'avenir le développement de techniques de réparations plus déterministes facilitera l'analyse des méthodes de recherche locale dans le but de proposer de nouvelles améliorations. Par ailleurs, nous avons montré que générer l'interprétation initiale par l'intermédiaire du déséquilibre des occurrences des variables ouvre de nouvelles pistes quant à la mise au point de nouvelles techniques de réparations locales.
- L'algorithme de Davis & Putnam est sans conteste l'approche complète la plus utilisée actuellement. Notre contribution à l'amélioration de cet algorithme porte sur la mise au point d'une nouvelle technique d'élagage de l'arbre DP basée sur une généralisation du théorème de partition du modèle. L'utilisation de cette technique au sein de la procédure de Davis & Putnam a permis de réduire sensiblement le nombre de nœuds parcourus par DP. L'étude théorique et pratique faite sur la généralisation du théorème de partition du modèle devrait permettre dans le futur d'intégrer notre technique d'élagage aux meilleures implantations de DP actuelles, afin d'accroître encore leur efficacité.
- Les approches mixtes sont jusqu'à l'heure actuelle peu connues et peu utilisées. Nous avons montré que de telles approches étaient tout à fait réalisables en pratique. En effet, nous avons proposé un nouvel algorithme complet, DPRL, utilisant la recherche locale à la fois comme heuristique de branchement de DP mais également comme un moyen de prolonger l'interprétation partielle courante construite par DP vers un modèle. Cette technique repose sur le constat que les méthodes de recherche locale, bien que dédiées uniquement à la recherche de solutions, permettent de circonscrire l'inconsistance de certaines instances de **SAT**. De plus, notre algorithme conserve toute l'efficacité des méthodes de recherche locale dans la preuve de l'existence d'un modèle et est particulièrement efficace sur les instances où l'inconsistance est locale, c'est-à-dire due à un sous-ensemble de l'ensemble de clauses. Les algorithmes mixtes constituent, à notre avis, une voie de recherche porteuse quant à la mise au point des algorithmes performants à venir.
- La génération d'instances aléatoires fait actuellement l'objet de nombreuses études, notamment dans le but d'améliorer les meilleures bornes théoriques du seuil. Nous avons étudié ce modèle de généra-

tion standard en introduisant un nouveau paramètre : la probabilité de « positiver » un littéral. L'étude de ce modèle de génération a permis de mettre en avant la relation entre le pic de difficulté des instances aléatoires et la probabilité de générer un littéral positif. D'autre part, un phénomène surprenant quant à l'apparition des littéraux purs lors de la résolution de ces instances par la procédure de Davis & Putnam a été exhibé. En effet, la courbe représentant le nombre de littéraux purs en fonction du rapport nombre de clauses sur nombre de variables, admet deux extremums dont l'un est situé exactement à la valeur du seuil. Pour ce dernier, nous n'avons aucune intuition permettant d'expliquer ce phénomène.

En ce qui concerne l'étude des techniques de déduction à partir d'une base de connaissances, notre contribution porte sur l'introduction de nouvelles techniques à base de compilations logiques préservant l'équivalence.

- D'un point de vue théorique, nous avons généralisé le concept d'impliquants en introduisant les notions d'impliquants modulo une ou des théories. Cette généralisation a eté adaptée aux notions d'impliquants premiers et de couverture d'impliquants (premiers), pour aboutir à la notion de couverture traitable. Nous avons proposé trois cas particuliers de couverture traitable : les couvertures modulo une théorie, les couvertures de simplifications traitables et les couvertures d'hyper-impliquants.
- D'un point de vue pratique, nous avons proposé différentes techniques de compilations logiques préservant l'équivalence basées sur les différentes couvertures définies. Nous avons montré que ces approches de compilation permettent de réduire la taille et les temps de compilation. De plus, les approches mises au point sont génériques et possèdent la propriété d'être « *anytime* ». Les techniques basées sur des raisonnements à partir de théories (traitables) ouvrent clairement de nouvelles pistes à la fois dans la mise en œuvre de nouveaux algorithmes de compilations logiques mais également dans la mise en place de nouvelles stratégies de résolution du problème **SAT**.

Le travail amorcé dans cette thèse ouvre de nombreuses perspectives et interrogations. Nous proposons d'étendre nos travaux dans diverses directions :

Dans l'avenir, nous nous attacherons bien évidemment à améliorer les différents algorithmes proposés.

- TSAT est pour l'instant un algorithme capable de régler automatiquement la longueur de la liste tabou pour des instances *k***SAT** aléatoires. Il reste que seul un réglage manuel de cette longueur est opéré actuellement pour les autres instances. Les travaux engagés sur un réglage dynamique de la liste tabou sont encourageants. Des résultats partiels montrent la convergence de la longueur de la liste tabou vers la taille optimale pour les instances *k***SAT** aléatoires. Cependant, ce réglage automatique ne se fait pas sans un coût supplémentaire, il nous faut donc déterminer un procédé permettant de garantir l'obtention de la taille optimale tout en conservant ou en améliorant les performances actuelles. L'une des voies aboutissant à cet objectif passe peut être par la prise en compte du déséquilibre des variables dans la stratégie de réparation de TSAT.
- L'application du théorème de partition du modèle dans la procédure de Davis & Putnam permet un gain substantiel en terme du nombre de nœuds explorés. Il reste que le temps de calcul joue en défaveur de cette technique. Plusieurs pistes sont actuellement en cours d'exploration pour diminuer le temps de calcul, notamment la détermination d'un ordre d'examen des clauses susceptibles d'être impliquées. Par ailleurs, les niveaux d'implication de clauses introduits en fin de thèse devraient permettre d'élaguer l'arbre de recherche construit par DP de manière encore plus significative et ainsi de réduire le temps de calcul de la méthode.
- DPRL est une technique mise au point uniquement pour montrer la faisabilité des méthodes mixtes. De nombreuses optimisations devraient permettre d'accroître très fortement ses performances. Nous pensons notamment limiter le nombre d'appels à la recherche locale et, d'autre part, intégrer le score des littéraux calculés par la recherche locale dans une heuristique de branchement plus sophistiquée tendant à équilibrer l'arbre de décision construit par la méthode.

– Les algorithmes de compilation logique proposés dans cette thèse sont tous basés sur la procédure de Davis & Putnam. Nous pensons qu'une voie pour améliorer les performances de ces algorithmes est l'utilisation d'heuristiques de branchement spécialisées. Par exemple, lors du calcul d'une couverture simplifications traitables, il serait intéressant de déterminer dans quelle mesure il est possible de déterminer une heuristique facilitant l'apparition de formules traitables. D'autre part, il est évident que seul un ensemble restreint de classes traitables ont été considérées, une amélioration possible des différentes techniques serait de mettre en jeu un plus large panel de classes traitables.

Certains résultats empiriques obtenus au cours de cette thèse posent un certain nombre d'interrogations qui ne trouveront réponses qu'après une analyse théorique poussée.

– La courbe obtenue sur le réglage de la taille de la liste d'interdits pour TSAT, mérite sans aucun doute une analyse théorique afin de savoir s'il est possible de lier cette taille aux paysages des instances *k***SAT**.
– L'importance des littéraux retournés par les méthodes de recherche locale (littéraux étant apparus le plus souvent dans des clauses falsifiées) étant démontrée expérimentalement, est-il possible de caractériser ces littéraux autrement que par l'exécution d'une méthode de recherche locale ?
– Nous avons montré expérimentalement le lien entre le pic de difficulté des instances *k***SAT** aléatoires et la probabilité de positiver un littéral dans ces instances. Est-il possible de caractériser ce lien de manière théorique ?

Une perspective intéressante est de voir dans quelle mesure ces résultats peuvent être étendus et appliqués à d'autres domaines. Un travail préliminaire (ne figurant pas dans cette thèse) a été entrepris quant à l'extension des techniques mises au point pour la satisfaisabilité à un contexte de logique non monotone et de diagnostic de pannes. Pour ce faire, nous nous sommes restreints à l'utilisation d'une logique propositionnelle non monotone, où certaines propositions doivent être minimisées (propositions d'anormalités à la McCarthy). Nous avons montré comment utiliser les techniques développées pour le problème **SAT**, en vue de rechercher des modèles préférés au sein de cette logique [Mazure *et al.* 1997d, Mazure *et al.* 1997a]. Par ailleurs nous avons proposé une nouvelle approche pour calculer de manière effective des inférences au sein d'une logique non monotone propositionnelle simple, fondée sur ce concept de modèles préférés [Grégoire & Mazure 1998, Grégoire *et al.* 1998].

Par ailleurs, les résultats obtenus dans le cadre de la détection et de la localisation des inconsistances nous ont conduits à appliquer ces techniques afin de résoudre des problèmes liés à la fusion et à l'interaction de bases de connaissances dans un cadre de travail coopératif. Le travail porte sur le maintien de la cohérence lors de la fusion de connaissances exprimées dans un formalisme de logique propositionnelle [Mazure *et al.* 1996c, Mazure *et al.* 1997c].

Annexe

Instances de DIMACS : description

Nous décrivons succinctement dans cette annexe, les instances proposées au challenge de DIMACS [DIM1993].

Ces informations ont été recueillies à partir du site internet de DIMACS :
`http://dimacs.rutgers.edu/Challenges/` .

L'ensemble de ces instances est accessible via un ftp anonyme à l'adresse :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/`

FAW : Ces instances ont été proposées par F.J. Radermacher and J. Mayer, elles proviennent d'horizons aussi variés que nombreux; on y trouve par exemple des instances du problème des pigeons, des problèmes de coloriage de graphe, des problèmes aléatoires, etc. Cependant, ces instances ont toutes la particularité d'être de « petite » taille.

SSA et BF : Les instances SSA et BF ont été proposées par Allen Van Gelder (`avg@cs.ucsd.edu`) et Yumi Tsuji (`tsuji@cse.ucsc.edu`). Elles représentent des problèmes d'analyse de pannes de circuits.

Dubois : Ces instances sont issues d'un générateur proposé par Olivier Dubois (`dubois@laforia.ibp.fr`). Ces instances ont la particularité de ne jamais produire de littéraux purs lors de leur résolution par DP et de fournir un arbre de taille exponentielle par rapport à la taille de l'instance (si toutefois aucune technique spécifique, comme la détection de symétries, n'est intégrée à DP). Le source du générateur est disponible via ftp anonyme à l'adresse :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/dubois/gensathard.c`

Par : Instances proposées par James Crawford (`jc@research.att.com`). Un problème de la forme parxx-y représente un problème de parité sur xx bits (y étant le numéro de l'instance) tandis que les problèmes parxx-y-c codent exactement le même problème excepté que ces instances ont été simplifiées de manière à créer un problème équivalent. De plus amples détails sur ces problèmes peuvent être trouvées à l'adresse :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/crawford/README`.

F : Proposés par Bart Selman (`selman@research.att.com`), ces problèmes sont des instances *3***SAT** aléatoires au seuil de « grande » taille (600, 1000 et 2000 variables).

G : Ces instances sont issues de problèmes de coloriage de graphes. Elles ont été proposées par Bart Selman (`selman@research.att.com`). Ces instances sont particulièrement difficiles car d'une part elles proviennent de problèmes de coloriage de graphes extrêmement difficiles à résoudre en théorie des graphes et d'autre part la représentation CNF de ces problèmes est très volumineuse (454622 clauses

pour le plus grand). De plus amples informations peuvent être obtenues à l'adresse :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/selman/README.cnf`.

Hanoi : Comme leur nom l'indique, ces problèmes représentent le codage CNF du problème des Tours de Hanoi. Ces problèmes ont été proposés par Bart Selman (`selman@research.att.com`).

Pret : Proposées par Daniele Pretolani (`daniele@crt.umontreal.ca`), ces instances codent des problèmes de 2-coloriage de graphes, enrichis par des contraintes de parité afin de forcer ces instances à être insatisfaisables. Le générateur de ces instances peut être trouvé à l'adresse :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/pretolani/`.

JNH : Ces problèmes sont des instances aléatoires proposées par John Hooker (`jh38+@andrew.cmu.edu`).

Hole : Problèmes des $n+1$ pigeons (ou chaussettes) et n pigeonniers (respectivement tiroirs). Vérifie s'il est possible de mettre $n+1$ pigeons dans n pigeonniers avec un seul pigeon par pigeonnier ! Ces instances ont été proposées par John Hooker (`jh38+@andrew.cmu.edu`).

AIM Ces instances proposées par Eiji Miyano (`miyano@csce.kyushu-u.ac.jp`) sont des instances *3***SAT** générées artificiellement de telle manière que les instances « *yes* » admettent un et un seul modèle. Le générateur de ces instances, ainsi que de plus amples informations sur ce générateur peuvent être trouvés dans le répertoire :
`ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/iwama/`.

Bibliographie

[AI 1996] *Special Volume on Frontier in Problem Solving: Phase Transitions and Complexity*, volume 81, numéro 1 de *Artificial Intelligence*, 1996.

[Anbulagan 1998] ANBULAGAN. « *Hypothèse de contrainte : une explication de la réussite de l' heuristique UP dans la résolution des problèmes de satisfiabilité des expressions booléennes* ». Thèse de doctorat, Université de Technologie de Compiègne, juin 1998.

[André 1993] Pascal ANDRÉ. « *Aspects probabilistes du problème de la satisfaction d'une formule booléenne. Étude des problèmes SAT, #SAT et max-SAT* ». Thèse de doctorat, Université Paris 6, 1993.

[Arvind & Biswas 1987] V. ARVIND et S. BISWAS. « An $\mathcal{O}(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes all Horn sentences ». *Information Processing Letters*, 24(1):67–69, 15 janvier 1987.

[Aspvall 1980] Bengt ASPVALL. « Recognizing Disguised NR(1) Instances of the Satisfiablity Problem ». *Journal of Algorithms*, 1:97–103, 1980.

[Aspvall *et al.* 1979] Bengt ASPVALL, Michael F. PLASS, et Robert Endre TARJAN. « A linear-time algorithm for testing the truth of certain quantified Boolean formulas ». *Information Processing Letters*, 8(3):121–123, mars 1979.

[Bailleux & Marquis 1999] Olivier BAILLEUX et Pierre MARQUIS. « DISTANCE-SAT: Complexity and Algorithms ». à paraître dans les actes de JNPC'99, 1999.

[Bailleux 1998] Olivier BAILLEUX. Communication personnelle, janvier 1998.

[Battiti & Protasi 1996] R. BATTITI et M. PROTASI. « Reactive Search, a history-based heuristic for MAX-SAT ». Technical report, Dipartimento di Mathematica, University of Trento (Italy), 1996.

[Bayardo Jr & Miranker 1996] R. J. BAYARDO JR et D. P. MIRANKER. « A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraint Satisfaction ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 558–562, juillet 1996.

[Bayardo Jr. & Schrag 1997] Roberto J. BAYARDO JR. et Robert C. SCHRAG. « Using CSP Look-Back Techniques to Solve Real-World SAT Instances ». Dans *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence (Rhode Island, USA), juillet 1997.

[BEI1996] « First International Competition and Symposium on Satisfiability Testing », mars 1996. Beijing (China).

[Belleannée & Vorc'h 1994] C. BELLEANNÉE et R. VORC'H. « A linear tableau proof for the pigeonhole formulae using symmetry ». Dans *Proceedings of the Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, Imperial College - London - UK Oxford, 1994. TR-94/5.

[Benhamou & Saïs 1992] Belaïd BENHAMOU et Lakhdar SAÏS. « Theoretical Study of Symmetries in Propositional Calculus and Applications ». Dans *Proceedings of Eleventh Interna-*

tional Conference on Automated Deduction (CADE-11), volume 607 de *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1992.

[Benhamou & Saïs 1994] Belaïd BENHAMOU et Lakhdar SAÏS. « Tractability Through Symmetries in Propositional Calculus ». *Journal of Automated Reasoning*, 12:89–102, 1994.

[Benhamou 1994] Belaïd BENHAMOU. « Study of Symmetry in constraint satisfaction problems ». Dans *Proceedings of PPCP'94*, Orcas Island Resario, Washington, 1994.

[Benhamou *et al.* 1994] Belaïd BENHAMOU, Lakhdar SAÏS, et Pierre SIEGEL. « Two Proof Procedures for a Cardinality Based Language in Propositional Calculus ». Dans P. ENJALBERT, E.W. MAYR, et K.W. WAGNER, éditeurs, *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 775 de *Lecture Notes in Computer Science*, pages 71–82, Caen (France), février 1994. Springer Verlag.

[Billionnet & Sutter 1992] Alain BILLIONNET et Alain SUTTER. « An efficient algorithm fot the 3-satisfiability problem ». *Operations Research Letters*, 12:29–36, 1992.

[Boole 1854] G. BOOLE. *Les lois de la pensée*. Mathesis, 1854.

[Borgida *et al.* 1989] A. BORGIDA, R.J. BRACHMAN, D.W. ETHERINGTON, et H.A. KAUTZ. « Vivid knowledge and tractable reasoning: Preliminary report ». Dans *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1146–1152, 1989.

[Boros *et al.* 1990] E. BOROS, Y. CRAMA, P.L. HAMMER, et M. SAKS. « A complexity index of satisfiability problems ». *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.

[Boros *et al.* 1994] E. BOROS, P.L. HAMMER, et X. SUN. « Recognition q-Horn formulae in linear time ». *Journal of Discrete Applied Mathematics*, 55:1–13, 1994.

[Boufkhad 1996] Yacine BOUFKHAD. « *Aspects probabilistes et algorithmiques du problème de satisfiabilité* ». Thèse de doctorat, Université de Paris 6, Laboratoire d' Informatique de Paris 6 (LIP6), décembre 1996.

[Boufkhad 1998] Yacine BOUFKHAD. « Algorithms for propositional KB approximation ». Dans *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 280–285, Madison (Wisconsin - USA), juillet 1998.

[Boufkhad *et al.* 1997] Yacine BOUFKHAD, Éric GRÉGOIRE, Pierre MARQUIS, Bertrand MAZURE, et Lakhdar SAÏS. « Tractable Cover Compilations ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 122–127, Nagoya (Japan), août 1997.

[Boy de la Tour & Demri 1995] T. Boy de la TOUR et S. DEMRI. « On the complexity of Extending Ground Resolution with Symmetry Rules ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 289–295, 1995.

[Brachman & Levesque 1982] Ronald J. BRACHMAN et Hector J. LEVESQUE. « Competence in Knowledge Representation ». Dans *Proceedings of the Second National Conference on Artificial Intelligence (AAAI'82)*, pages 189–192, 1982.

[Brisoux *et al.* 1998] Laure BRISOUX, Lakhdar SAÏS, et Éric GRÉGOIRE. « Mieux Exploiter les échecs au sein des arbres de recherche à la Davis et Putnam ». Dans *Actes des Quatrièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC'98)*, pages 31–39, Nantes (France), mai 1998.

[Cadoli & Donini 1999] Marco CADOLI et Francesco M. DONINI. « A survey on Knowledge Compilation ». à paraître dans AI Communications, 1999.

[Cadoli 1993] Marco CADOLI. « Semantical and computational aspects of Horn approximations ». Dans *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 39–44, 1993.

[Castell & Cayrol 1996a] Thierry CASTELL et Michel CAYROL. « Arguments for easy generation of random, hard and satisfiable instances of SAT (Easy to be hard II) ». Dans *Pro-*

ceedings of ECAI'96 Workshop on Advances in Propositional Deduction, pages 5–10, Budapest (Hungary), août 1996.

[Castell & Cayrol 1996b] Thierry CASTELL et Michel CAYROL. « Computation of prime implicates and prime implicants by the Davis and Putnam Procedure ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 61–64, Budapest (Hungary), août 1996.

[Castell 1996] Thierry CASTELL. « Résolution arrière dans la procédure de Davis et Putnam ». Dans *Rencontres Françaises en Intelligence Artificielle (RFIA'96)*, pages 109–117, Rennes (France), 1996.

[Castell 1997] Thierry CASTELL. « *Consistance et déduction en logique propositionnelle* ». Thèse de doctorat, Université Paul Sabatier, Toulouse, janvier 1997.

[Cha & Iwama 1995] Byungki CHA et Kazuo IWAMA. « Performance Test of Local Search Algorithms Using New Types of Random CNF Formulas ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 304–310, 1995.

[Chabrier 1997] Jacqueline CHABRIER. « *Programmation Par Contraintes: langages, méthodes et applications sur les domaines entiers et booléens* ». Thèse d'habilitation à diriger des recherches, Université de Bourgogne, décembre 1997.

[Chabrier *et al.* 1991] Jacqueline CHABRIER, Jean-Jacques CHABRIER, et F. TROUSSET. « Résolution Efficace d'un Problème de Satisfaction de Contraintes : le million de reines ». Dans *Onzièmes Journées Internationales sur les Systèmes Experts et leurs Applications*, 1991.

[Chabrier *et al.* 1995] Jacqueline CHABRIER, Vincent JULIARD, et Jean-Jacques CHABRIER. « SCORE(FD/B) an efficient complete local-based serach method for satisfiability problems ». Dans *Proceedings of Constraint Programming Workshop on Solving Really Hard Problems*, pages 25–30, Cassis (France), septembre 1995.

[Chabrier *et al.* 1996] Jacqueline CHABRIER, Jean-Michel RICHER, et Chabrier JEAN-JACQUES. « Resolution of structred SAT problems with SCORE(FD/B) ». Dans *Proceedings of the CP'96 Workshop on Constraint Programming Applications*, Boston, (MA, USA), août 1996.

[Chandra & Markowsky 1978] A.K. CHANDRA et G. MARKOWSKY. « On the number of prime implicants ». *Discrete Mathematics*, 2:7–11, 1978.

[Chandru & Hooker 1990] V. CHANDRU et J.N. HOOKER. « Extended Horn sets in Propositional Logic ». *Journal of the Association for Computing Machinery*, 38:205–221, 1990.

[Chandru *et al.* 1990] V. CHANDRU, C.R. COULARD, P.L. HAMMER, M. MONTANEZ, et X. SUN. « On renamable Horn and generalized Horn Fonctions ». Dans Scientific Publishing Company J.C. BALTZER AG, éditeur, *Annals of Mathematics and Artificial Intelligence*, volume 1, Basel (Switzerland), 1990.

[Chang & Lee 1973] Chin-Liang CHANG et Richard Char-Tung LEE. *Symbolic logic and mechanical theorem proving*. Academic Press, 1973.

[Cheeseman *et al.* 1991] Peter CHEESEMAN, Bob KANEFSKY, et William M. TAYLOR. « Where the Really Hard Problems Are ? ». Dans *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 331–337, 1991.

[Chvátal & Reed 1992] V. CHVÁTAL et B. REED. « Miks gets some (the odds are on this side) ». Dans *Proceedings of the 33^{rd} IEEE Symposium on Foundations of Computer Science*, pages 620–627, 1992.

[Chvátal & Szemerédi 1988] Vašek CHVÁTAL et Endre SZEMERÉDI. « Many hard examples for resolution ». *Journal of the Association for Computing Machinery*, 35(4):759–768, 1988.

[Conforti & Cornuéjols 1992] M. CONFORTI et G. CORNUÉJOLS. « A class of logical inference problems solvable by linear programming ». Dans *Proceedings of FOCS'92*, volume 33, pages 670–675, 1992.

[Cook 1971] S. A. COOK. « The complexity of theorem-proving procedures ». Dans *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York (USA), 1971. Association for Computing Machinery.

[Crawford & Auton 1993] James M. CRAWFORD et Larry D. AUTON. « Experimental Results on the Crossover Point in Satisfiablity Problems ». Dans *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 21–27, 1993.

[Crawford & Auton 1996] James M. CRAWFORD et Larry D. AUTON. « Experimental Results on the Crossover Point in Random 3SAT ». *Artificial Intelligence*, 81(1-2), 1996.

[Crawford 1992] James M. CRAWFORD. « A theorical analysis of reasoning by symmetry in first-order logic ». Dans *Proceedings of the AAAI'92 Workshop on Tractable Reasoning*, San Jose (USA), juillet 1992.

[Crawford 1993] James M. CRAWFORD. « Solving Satisfiability Problems Using a Combination of Systematic and Local Search ». Dans *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[Cubbada & Mouseigne 1988] C. CUBBADA et M.D. MOUSEIGNE. « *Variantes de l'algorithme de SL-résolution avec retenue d'information* ». Thèse de doctorat, Université de Provence (GIA Luminy), Marseille (France), 1988.

[Dalal & Etherington 1992] Mukesh DALAL et David W. ETHERINGTON. « A hierarchy of tractable satisfiability problems ». *Information Processing Letters*, 44(4):173–180, 10 décembre 1992.

[Dalal 1992] Mukesh DALAL. « Efficient Propositional Constraint Propagation ». Dans *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 409–414, San-Jose, California (USA), 1992.

[Dalal 1995] Mukesh DALAL. « *Tractable Reasoning in Knowledge Representations Systems* ». PhD thesis, Graduate School, New Brunswick, New Jersey (USA), mai 1995.

[Dalal 1996] Mukesh DALAL. « An Almost Quadratic Class of Satisfiability Problems ». Dans W. WAHLSTER, éditeur, *Proceedings of the Twelfth European Conference on Artificial Intelligence(ECAI'96)*, pages 355–359, Budapest (Hungary), août 1996. John Wiley & Sons, Ltd.

[Davis & Putnam 1960] Martin DAVIS et Hilary PUTNAM. « A Computing Procedure for Quantification Theory ». *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Davis *et al.* 1962] Martin DAVIS, George LOGEMANN, et Donald LOVELAND. « A Machine Program for Theorem Proving ». *Journal of the Association for Computing Machinery*, 5:394–397, 1962.

[De Kleer 1986] J. DE KLEER. « An Assumption-Based Truth Maintenance System ». *Artificial Intelligence*, 28:127–162, 1986.

[De Kleer 1992] J. DE KLEER. « An improved incremental algorithm for generating prime implicates ». Dans *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 780–785, 1992.

[Dechter & Rish 1994] Rina DECHTER et Irina RISH. « Directional Resolution: the Davis-Putnam procedure revisited ». Dans J. DOYLE, E. SANDEWALL, et P. TORASSI, éditeurs, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 134–145, 1994.

[Del Val 1994] Alvaro DEL VAL. « Tractable Databases: How to make propositional unit resolution complete through compilation ». Dans J. DOYLE, E. SANDEWALL, et P. TORASSI, éditeurs, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994.

[Del Val 1995] Alvaro DEL VAL. « An analysis of approximate knowledge compilation ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 830–836, 1995.

[Del Val 1996] Alvaro DEL VAL. « Approximate knowledge compilation: The first order case ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 498–503, juillet 1996.

[DIM1993] « Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University », 1993. `http://dimacs.rutgers.edu/Challenges/`.

[Dowling & Gallier 1984] 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.

[Dubois & Boufkhad 1996] Olivier DUBOIS et Yacine BOUFKHAD. « From very hard doubly balanced SAT formulae to easy unbalanced SAT formulae, variations of the satisfiability threshold ». Dans J.G. DING-ZHU DU et P. PARDALOS, éditeurs, *Proceedings of the DIMACS Workshop on the Satisfiability Problem: Theory and Applications*, mars 1996.

[Dubois & Boufkhad 1997] Olivier DUBOIS et Yacine BOUFKHAD. « A General Upper Bound for the Satisfiability Threshold of Random r-SAT Formulae ». *Journal of Algorithms*, 24(2):395–420, août 1997.

[Dubois & Carlier 1991] Olivier DUBOIS et Jacques CARLIER. « Probabilistic Approach to the Satisfiability Problem ». *Theoretical Computer Science*, 81:65–75, 1991.

[Dubois 1990] Olivier DUBOIS. « On the r,s-SAT satisfiability problem and a conjecture of Tovey ». *Discrete Applied Mathematics*, 26:51–60, 1990.

[Dubois *et al.* 1996] Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD, et Jacques CARLIER. « SAT versus UNSAT ». Dans D.S. JOHNSON et M.A. TRICK, éditeurs, *Second DIMACS Challenge*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, pages 415–436, 1996.

[Escalada-Imaz 1989] G. ESCALADA-IMAZ. « *Optimisation d'algorithmes d'inférence monotone en logique des propositions et du premier ordre* ». Thèse de doctorat, Université Paul Sabatier, Toulouse (France), 1989.

[Even *et al.* 1976] S. EVEN, A. ITAI, et A. SHAMIR. « On the Complexity of Timetable and Multicommodity Flow Problems ». *SIAM J. Comput.*, 5:691–703, 1976.

[Forbus & De Kleer 1993] K.D. FORBUS et J. DE KLEER. *Building Problem Solvers*. MIT Press, 1993.

[Franco 1991] John FRANCO. « Elimination of infrequent varaibles improves average case performance of satisfiability algorithms ». *SIAM Journal on Computing*, 20(6):1119–1127, décembre 1991.

[Freeman 1995] Jon William FREEMAN. « *Improvements to Propositional Satisfiability Search Algorithms* ». PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1995.

[Freeman 1996] Jon William FREEMAN. « Hard Random 3-SAT Problems and the Davis-Putnam Procedure ». *Artificial Intelligence*, 81(1-2):183–198, 1996.

[Freuder *et al.* 1995] Eugene C. FREUDER, Rina DECHTER, Matthew L. GINSBERG, Bart SELMAN, et E. TSANG. « Systematic Versus Stochastic Constraint Satisfaction ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, volume 2, pages 2027–2032, 1995.

[Frieze & Suen 1996] Alan FRIEZE et Stephen SUEN. « Analysis of Two Simple Heuristics on a Random Instance of k-SAT ». *Journal of Algorithms*, 20(2):312–355, mars 1996.

[Galil 1977] Z. GALIL. « On the complexity of regular resolution and the Davis-Putnam Procedure ». *Theoretical Computer Science*, 4:23–46, 1977.

[Gallo & Scutellá 1988] Giorgio GALLO et Maria Grazia SCUTELLÁ. « Polynomially solvable satisfiability problems ». *Information Processing Letters*, 29(5):221–227, 24 novembre 1988.

[Gallo & Urbani 1989] Giorgio Gallo GALLO et Giampaolo URBANI. « Algorithms for Testing the Satisfiability of Propositional Formulae. ». *Journal of Logic Programming*, 7(1):45–61, juillet 1989.

[Garey & Johnson 1979] Michael R. GAREY et David S. JOHNSON. *Computers and Intractability: A Guide to the Theory on NP-completness*. A series of books in the Mathematical Sciences. W. H. Freeman and Company, New-York (USA), 1979.

[Génisson & Rauzy 1996] Richard GÉNISSON et Antoine RAUZY. « Sur les relations algorithmiques des classes polynomiales du problème SAT et des problèmes de satisfaction de contraintes ». Dans *Actes de RFIA'96*, pages 97–107, 1996.

[Génisson & Saïs 1994] Richard GÉNISSON et Lakhdar SAÏS. « Some ideas on random generation of K-SAT instances ». Dans *post-conference workshop on Experimental Evaluation of Reasoning and Search Methods, in Proceedings of AAAI'94*, pages 91–92, 1994.

[Génisson & Siegel 1994] Richard GÉNISSON et Pierre SIEGEL. « A polynomial method for sub-clauses production ». Dans *Proceedings of Sixth International Conference on Artificila Intelligence: Methodology, Systems, Applications (AIMSA'94)*, pages 25–34, North Holland, 1994.

[Gent & Walsh 1993a] Ian P. GENT et Tobby WALSH. « An Empirical Analysis of Search in GSAT ». *Journal of Artificial Intelligence Research (JAIR)*, 1:47–59, 1993.

[Gent & Walsh 1993b] Ian P. GENT et Tobby WALSH. « Towards an Understanding of Hill-Climbing Procedures of SAT ». Dans *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 28–33, 1993.

[Gent & Walsh 1994a] Ian P. GENT et Tobby WALSH. « Easy problems are sometimes hard ». *Artificial Intelligence*, 70:335–345, 1994.

[Gent & Walsh 1994b] Ian P. GENT et Tobby WALSH. « The SAT phase transition ». Dans *Proceedings of the Eleventh European Conference on Artificial Intelligence(ECAI'94)*, pages 105–109, 1994.

[Ghallab & Escalada-Imaz 1991] M. GHALLAB et E. ESCALADA-IMAZ. « A linear control algorithm for a class of rule-based systems ». *Journal of Logic Programming*, 11:117–132, 1991.

[Ginsberg & McAllester 1994] Matthew L. GINSBERG et David A. MCALLESTER. « GSAT and Dynamic Backtracking ». Dans J. DOYLE, E. SANDEWALL, et P. TORASSI, éditeurs, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 226–236, 1994.

[Ginsberg 1993] Matthew L. GINSBERG. « Dynamic Backtracking ». *Journal of Artificial Intelligence Research (JAIR)*, 1:25–46, 1993.

[Glover & Laguna 1993] F. GLOVER et M. LAGUNA. « *Tabu Search* », pages 70–141. Oxford: Blackwell Scientific, 1993.

[Glover 1989] F. GLOVER. « Tabu search - Part I ». *ORSA Journal of Computing*, 1:190–206, 1989.

[Glover 1990] F. GLOVER. « Tabu search - Part II ». *ORSA Journal of Computing*, 2:4–32, 1990.

[Glover *et al.* 1993] F. GLOVER, E. TAILLARD, et D. WERRA. « A User's Guide to Tabu Search ». *Annals of operations Research*, 41:3–28, 1993.

[Gogic *et al.* 1994] G. GOGIC, C.H. PAPADIMITRIOU, et M. SIDERI. « Incremental recompilation of knowledge ». Dans *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 922–927, 1994.

[Goldberg 1989] D.E. GOLDBERG. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Grégoire & Mazure 1998] Éric GRÉGOIRE et Bertrand MAZURE. « Une méthode complète de recherche locale pour des bases de connaissances propositionnelles non monotones ». Dans *Actes des Quatrièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (JNPC'98)*, pages 91–99, Nantes (France), mai 1998.

[Grégoire *et al.* 1998] Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Logically-complete local search for propositional nonmonotonic knowledge bases ». Dans Ilkka

NIEMELÄ et Torsten SCHAUB, éditeurs, *Proceedings of the KR'98 Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 37–45, Trento, 1998. Helsinky University of Technology Research Report, Digital Systems Laboratory.

[Gu 1992] Jun GU. « Efficient local search for very large-scale satisfiability problems ». *SIGART Bulletin*, 3(1):8–12, janvier 1992.

[Hall 1935] P. HALL. « On representatives of subsets ». *J. London Math. Society*, 10:26–30, 1935.

[Hansen & Jaumard 1990] Pierre HANSEN et Brigitte JAUMARD. « Algorithms for the Maximum Satisfiability Problem ». *Journal of Computing*, 22:279–303, 1990.

[Hao & Tétard 1996] Jin-Kao HAO et Laurent TÉTARD. « CH-SAT: A complete Heuristic procedure for satisfiability problem ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 27–38, Budapest (Hungary), août 1996.

[Hébrard 1994] Jean-Jacques HÉBRARD. « A linear algorithm for renaming a set of clauses as a Horn set ». *Theoretical Computer Science*, 124:343–350, 1994.

[Hébrard 1995] Jean-Jacques HÉBRARD. Base de Horn d'un ensemble de clauses. Dans *Les cahiers de greyc*, numéro 2 dans GREYC. Université de Caen, France, 1995.

[Hooker & Vinay 1995] J.N. HOOKER et N. VINAY. « Branching rules for satisfiability ». *Journal of Automated Reasoning*, 15:359–383, 1995.

[Hooker 1993] J.N. HOOKER. « Solving the Incremental Satisfiability Problem ». *Journal of Logic Programming*, 15:177–186, 1993.

[Humbert 1995] E. HUMBERT. « *Études de complexité au pire dans le problème de satisfiabilité logique* ». PhD thesis, Université René Descartes – Paris V, novembre 1995.

[Jackson & Pais 1990] P. JACKSON et J. PAIS. « Computing prime implicants ». Dans *Proceedings of Tenth International Conference on Automated Deduction (CADE-10)*, pages 543–557, 1990.

[Jaumard *et al.* 1993] Brigitte JAUMARD, Mihnea STAN, et Jacques DESROSIERS. « Tabu Search and a quadratic relaxation for the satisfiability problem ». Dans *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[Jeannicot *et al.* 1988] S. JEANNICOT, Laurent OXUSOFF, et Antoine RAUZY. « Évaluation sémantique en calcul propositionnel : une propriété de coupure pour rendre plus efficace la procédure de Davis et Putnam ». *Revue d'Intelligence Artificielle*, 2(1):41–60, 1988.

[Jeroslow & Wang 1990] Robert G. JEROSLOW et Jinchang WANG. « Solving Propositional Satisfiability Problems ». *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Kask & Dechter 1995] K. KASK et R. DECHTER. « GSAT and Local Consistency ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 616–622, 1995.

[Kask & Dechter 1996] K. KASK et R. DECHTER. « A Graph-Based Method for Improving GSAT ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 350–355, juillet 1996.

[Kautz & Selman 1992] Henry A. KAUTZ et Bart SELMAN. « Forming concepts for fast inference ». Dans *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 786–793, 1992.

[Kirkpatrick *et al.* 1983] S. KIRKPATRICK, C.D. GELLAT, et M.P. VECHI. « Optimization by Simulated Annealing ». *Science*, 220(4598):671–680, 1983.

[Kirousis *et al.* 1998] Lefteris M. KIROUSIS, Evangelos KRANAKIS, Danny KRIZANC, et Yannis C. STAMATIOU. « Approximating the unsatisfiability threshold of random formulas ». *Journal of Random Structures & Algorithms*, 12(3):253–269, 1998. Un « extended abstract » de ce papier est paru dans « Proceedings of the Fourth Annual European Symposium on Algorithms », ESA'9l, 25-27 Septembre 1996, Barcelone, Espagne (Springer Verlag, LNCS, pp. 27-38).

[Knuth 1990] Donald E. KNUTH. « Nested Satisfiability ». *Acta Informatica*, 28:1–6, 1990.

[Kowalski & Kuehner 1971] R.A. KOWALSKI et D. KUEHNER. « Linear Resolution with Selection Function ». *Artificial Intelligence*, 2:227–260, 1971.

[Kratochvtil *et al.* 1993] J. KRATOCHVTIL, P. SAVICKY, et Z. TUZA. « One more occurrence of variable makes satisfiability jump from trivial to NP-complete ». *SIAM Journal of Computing*, 22:203–210, 1993.

[Krishnamurthy 1982] B. KRISHNAMURTHY. « *Examples of hard tautologies and Word-Case complexity results in propositional calculus* ». PhD thesis, Dept. of Computer and Information Science, University of Massachussets, 1982.

[Krishnamurthy 1985] B. KRISHNAMURTHY. « Short Proofs for Tricky Formulas ». *Acta Informatica*, 22:253–275, 1985.

[Lacoste 1996] Thierry LACOSTE. « *Lois de Convergence et Lois 0-1 dans les Structures Aléatoires Finies : une approche Logique et Finitiste* ». Thèse de doctorat, Université Paris 7 Denis Diderot, juin 1996.

[Larrabee & Tusji 1993] T. LARRABEE et Y. TUSJI. « Evidence for satisfiability threshold for rand 3CNF formalas ». Dans H.E. HIRSH, éditeur, *Proceedings of Spring Symposium on Artificial Intelligence and NP-Hard Problems*, pages 112–118, Stanford (CA, USA), 1993.

[Levesque 1986] Hector J. LEVESQUE. « Making believers out of computers ». *Artificial Intelligence*, 30:81–108, 1986.

[Lewis 1978] H.R. LEWIS. « Renaming a Set of Clauses as Horn Set ». *Journal of the Association for Computing Machinery*, 25:134–135, 1978.

[Li & Anbulagan 1997] Chu Min LI et ANBULAGAN. « Heuristics Based on Unit Propagation for Satisfiability Problems ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya (Japan), août 1997.

[Li 1996] Chu Min LI. « Exploiting yet more the power of unit clause propagation to solve 3-SAT problem ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 11–16, Budapest (Hungary), août 1996.

[Lichtenstein 1982] David LICHTENSTEIN. « Planar formulae and their uses ». *SIAM Journal of Computing*, 11(2):329–343, mai 1982.

[Lobjois & Lemaître 1997] Lionel LOBJOIS et Michel LEMAÎTRE. « Coopération entre méthodes complètes et incomplètes pour la résolution de (V)CSP : une tentative d'inventaire ». Dans *Actes des Troisièmes Journées Nationales sur la Résolution Pratique de Problèmes NPè-complets (JNPC'97)*, pages 67–73, Rennes (France), avril 1997. PRC-GDR IA – Universitè de Rennes 1.

[Lozinskii 1993] Eliezer L. LOZINSKII. « A simple test improves checking satisfiability ». *Journal of Logic Programming*, 15:99–111, 1993.

[Madre & Coudert 1991] J.C. MADRE et O. COUDERT. « A logically complete reasoning maintenance system based on a logical constraint solver ». Dans *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 294–299, 1991.

[Marquis & Sadaoui 1996] Pierre MARQUIS et Samira SADAOUI. « A new algorithm for computing theory prime implicates compilations ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 504–509, juillet 1996.

[Marquis 1995a] Pierre MARQUIS. « From theory prime implicates compilations to minimal implicates compilations (extended abstract) ». Dans *Proceedings of KI'95 Workshop on Computational Propositional Logic (CLP'95)*, pages 85–86, 1995.

[Marquis 1995b] Pierre MARQUIS. « Knowledge compilation using theory prime implicates ». Dans *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 837–843, 1995.

[Marquis 1997] Pierre MARQUIS. Communication personnelle, novembre 1997.

[Mathieu & Delahaye 1990] Phillipe MATHIEU et Jean-Paul DELAHAYE. « The logical compilation of knowledge bases ». Dans *Proceedings of JELIA'90*, volume 478 de *Lecture Notes in Artificial Intelligence*, pages 386–398, Amsterdam (Holland), 1990. Springer Verlag.

[Mathieu & Delahaye 1994] Phillipe MATHIEU et Jean-Paul DELAHAYE. « A kind of logical compilation for knowledge bases ». *Theoretical Computer Science*, 131:197–218, 1994.

[Mazure & Marquis 1996] Bertrand MAZURE et Pierre MARQUIS. « Theory Reasoning within Implicant Cover Compilations ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 65–69, Budapest (Hungary), août 1996.

[Mazure *et al.* 1995] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « TWSAT: a new local search algorithm for SAT. Performance and Analysis ». Dans *Proceedings of Constraint Programming Workshop on Solving Really Hard Problems*, pages 127–130, Cassis (France), septembre 1995.

[Mazure *et al.* 1996a] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Detecting Logical Inconsistencies (Extended Abstract) ». Dans *Proceedings of Mathematics and Artificial Intelligence Symposium*, pages 116–121, Fort Lauderdale (FL USA), janvier 1996.

[Mazure *et al.* 1996b] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Deux approches pour la résolution du problème SAT ». Dans *Actes des Deuxièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets (CNPC'96)*, pages 103–114, Dijon (France), mars 1996.

[Mazure *et al.* 1996c] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « A Powerful Heuristic to locate Inconsistent Kernels in Kwnowledge-Based Systems ». Dans *Proceedings of IPMU'96*, volume 3, pages 1265–1269, Granada (Spain), juillet 1996.

[Mazure *et al.* 1996d] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « SUn: A multistrategy Platform for SAT ». Dans *Working notes of the First International Competition and Symposium on Satisfiability Testing*, Beijing (China), mars 1996.

[Mazure *et al.* 1997a] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Checking Several Forms of Consistency in Non-Monotonic Knowledge-Bases ». Dans Dov GABBAY, Rudolf KRUSE, Andreas NONNENGART, et Hans Jüergen OHLBACH, éditeurs, *Proceedings of the First International Joint Conference on Qualitative and Quantitative Pratical Reasoning (ECSQARU-FAPR'97)*, volume 1244 de *Lecture Notes in Artificial Intelligence*, pages 122–130, Bad Honnef (Germany), juin 1997. Springer.

[Mazure *et al.* 1997b] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « A comparison of Two Approaches to Inconsistency Detecting ». Dans *European Symposium on Intelligent Techniques*, Bari (Italy), mars 1997.

[Mazure *et al.* 1997c] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « An Efficient Technique to ensure the Logical Consistency of Interacting Knowledge Bases ». *International Journal of Cooperative Information Systems*, 6(1):27–36, 1997.

[Mazure *et al.* 1997d] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Local Search for Computing Normal Circumstances Models (Abstract) ». Dans B. REUSCH, éditeur, *Proccedings of the Computational Intelligence Conference (Theory and Applications)*, volume 1226 de *Lecture Notes in Computer Science*, pages 565–566, Dortmund (Germany), avril 1997.

[Mazure *et al.* 1997e] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Tabu Search for SAT ». Dans *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, Providence (Rhode Island, USA), juillet 1997.

[Mazure *et al.* 1998a] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Boosting Complete Techniques thanks to local search methods ». *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.

[Mazure *et al.* 1998b] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « System Description: CRIL Platform for SAT ». Dans *Proceedings of the Fifteenth International Conference on Automated Deduction (CADE'15)*, volume 1421 de *Lecture Notes in Artificial Intelligence*, pages 116–119, Lindau (Germany), juillet 1998.

[McAllester *et al.* 1997] David A. MCALLESTER, Bart SELMAN, et Henry A. KAUTZ. « Evidence for Invariants in Local Search ». Dans *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, août 1997.

[Minoux 1988] 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.

[Minton *et al.* 1990] Steven MINTON, Marc D. JOHNSTON, Andrew B. PHILIPS, et Philip LAIRD. « Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method ». Dans *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI'90)*, 1990.

[Mitchell *et al.* 1992] David MITCHELL, Bart SELMAN, et Hector J. LEVESQUE. « Hard and Easy Distributions of SAT Problems ». Dans *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.

[Morris 1993] P. MORRIS. « The Break Out Method For Escaping From Local Minima ». Dans *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, pages 40–45, 1993.

[Ngair 1993] T.-H. NGAIR. « A new algorithm for incremental prime implicate generation ». Dans *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 46–51, 1993.

[Oxusoff & Rauzy 1989] Laurent OXUSOFF et Antoine RAUZY. « *L'évaluation sémantique en calcul propositionnel* ». Thèse de doctorat, Université de Provence, GIA–Luminy, Marseille (France), janvier 1989.

[Papadimitriou 1991] Christos H. PAPADIMITRIOU. « On selecting a satisfying truth assignment ». Dans *Proceedings of the* 32^{rd} *Symposium on the Foundations of Computer Science*, pages 163–169, 1991.

[Papadimitriou 1994] Christos H. PAPADIMITRIOU. *Computational Complexity*. Addison-Wesley, 1994.

[Parkes & Walser 1996] Andrew J. PARKES et Joachim P. WALSER. « Tuning Local Search for Satisfiability Testing ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 356–362, juillet 1996.

[Pretolani 1993] D. PRETOLANI. « *Satisfiability and Hypergraphs* ». PhD thesis, dipartimento di Informatica: Universita di Pisa, Genova, Italia, mars 1993. TD-12/93.

[Rauzy 1994] Antoine RAUZY. « On the Complexity of the Davis & Putnam's Procedure on Some Polynomial Sub-Classes of SAT ». Rapport Interne 806-94, LaBRI, Université de Bordeaux I, 1994.

[Rauzy 1995a] Antoine RAUZY. « On the Random Generation of 3-SAT Instances ». Rapport Interne 1060-95, LaBRI, Université de Bordeaux I, 1995.

[Rauzy 1995b] Antoine RAUZY. « Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure. ». Dans U. MONTANARI et F. ROSSI, éditeurs, *Proceedings of the International Conference of Principle of Constraint Programming, CP'95*, volume 976 de *Lecture Notes in Computer Science*, pages 515–532. Springer Verlag, 1995.

[Reiter & De Kleer 1987] R. REITER et J. DE KLEER. « Foundations of assumption-based truth maintenance systems: Preliminary report ». Dans *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 183–188, 1987.

[Robinson 1963] J.A. ROBINSON. « Theorem proving on computer ». *Journal of the Association for Computing Machinery*, pages 163–174, 1963.

[Robinson 1965] J.A. ROBINSON. « A machine-oriented logic based on the resolution principle ». *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

[Robinson 1983] J.A. ROBINSON. Automatic deduction with hyperresolution. Dans *Automation of Reasoning-Classical Papers on Computational Logic*, volume 1 and 2. Springer, 1983.

[Rossa 1993] P. ROSSA. « Formules bien imbriquées : reconnaissance et satisfaisabilité ». Mémoire de DEA, Université de Caen, Laboratoire d'Informatique, 1993.

[Roth 1996] Dan ROTH. « On the hardness of approximate reasoning ». *Artificial Intelligence*, 82:273–302, 1996.

[Roussel & Mathieu 1996a] Olivier ROUSSEL et Phillipe MATHIEU. « How to Use Cycles for Logical Compilation ». Dans *Proceedings of ECAI'96 Workshop on Advances in Propositional Deduction*, pages 53–60, Budapest (Hungary), août 1996.

[Roussel & Mathieu 1996b] Olivier ROUSSEL et Phillipe MATHIEU. « A New Method for Knowledge Compilation: the Achievement by Cycle Search ». Dans M.A. MCROBBIE et J.K. SLANEY, éditeurs, *Proceedings of Thirteenth International Conference on Automated Deduction (CADE-13)*, volume 1104 de *Lecture Notes in Artificial Intelligence*, pages 493–507. Springer Verlag, 1996.

[Roussel 1997] Olivier ROUSSEL. « *L'achèvement des bases de connaissances en calcul propositionnel et en calcul des prédicats* ». Thèse de doctorat, Univerité des Sciences et Technologies de Lille (USTL-Lille 1), Laboratoire d' Informatique Fondamentale de Lille (LIFL), Villeneuve d'Ascq, janvier 1997.

[Saïs 1993] Lakhdar SAÏS. « *Étude des Symétries et de la Cardinalité en Calcul Propositionnel : Application aux algorithmes sémantiques* ». Thèse de doctorat, Université de Provence, GIA–Luminy, Marseille (France), février 1993.

[Schlipf *et al.* 1995] John S. SCHLIPF, Fred S. ANNEXSTEIN, John V. FRANCO, et R. P. SWAMINATHAN. « On finding solutions for extended Horn formulas ». *Information Processing Letters*, 54(3):133–137, 12 mai 1995.

[Schrag & Crawford 1996] Robert C. SCHRAG et James M. CRAWFORD. « Implicates and prime implicates in random 3SAT ». *Artificial Intelligence*, 81:199–222, 1996.

[Schrag 1996] Robert C. SCHRAG. « Compilation for critically constrained knowledge bases ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 510–515, juillet 1996.

[Scutellà 1990] Maria Grazia SCUTELLÀ. « A Note on Dowling and Gallier's Top Down Algorithm for Propositional Horn Satisfiablity ». *Journal of Logic Programming*, 8:265–273, 1990.

[Selman & Kautz 1991] Bart SELMAN et Henry A. KAUTZ. « Knowledge compilation using Horn approximations ». Dans *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 904–909, 1991.

[Selman & Kautz 1993a] Bart SELMAN et Henry A. KAUTZ. « Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems ». Dans *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295, 1993.

[Selman & Kautz 1993b] Bart SELMAN et Henry A. KAUTZ. « An Empirical Study of Greedy Local Search for Satisfiability Testing ». Dans *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, 1993.

[Selman & Kautz 1994] Bart SELMAN et Henry A. KAUTZ. « Knowledge Compilation and Theory Approximation ». *Journal of the Association for Computing Machinery*, 43(2):193–224, 1994.

[Selman & Kautz 1995] Bart SELMAN et Henry A. KAUTZ. « *GSAT USER'S GUIDE* ». AT&T Bell Laboraties, juillet 1995. GSAT version 41.

[Selman *et al.* 1992] Bart SELMAN, Hector J. LEVESQUE, et David MITCHELL. « GSAT: A New Method for Solving Hard Satisfiability Problems ». Dans *Proceedings of the*

Tenth National Conference on Artificial Intelligence (AAAI'92), pages 440–446, 1992.

[Selman *et al.* 1993] Bart SELMAN, Henry A. KAUTZ, et B. COHEN. « Local Search Strategies for Satisfiability Testing ». Dans *Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

[Selman *et al.* 1994] Bart SELMAN, Henry A. KAUTZ, et B. COHEN. « Noise strategies for improving local search ». Dans *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, 1994.

[Selman *et al.* 1997] Bart SELMAN, Henry A. KAUTZ, et David A. MCALLESTER. « Computational Challenges in Propositional Reasoning and Search ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 50–54, Nagoya (Japan), août 1997.

[Siegel 1987] Pierre SIEGEL. « *Représentation et Utilisation de la connaissance en calcul propositionnel* ». Thèse d'état, Université de Provence, GIA–Luminy, Marseille (France), 1987.

[Silva & Sakallah 1996a] J.P.M. SILVA et K.A. SAKALLAH. « Conflict Analysis in Search Algorithms for Propositional Satisfiability ». Rapport Technique RT/4/96, Grupo ALGOS: Instituto de Engenharia de Sistemas e Computadores, Lisboa (Portugal), mai 1996.

[Silva & Sakallah 1996b] J.P.M. SILVA et K.A. SAKALLAH. « GRASP - A New Search Algorithm for Satisfiability ». Dans *Proceedings of International Conference on Computer Aided Design*, 1996.

[Slagle *et al.* 1970] J.R. SLAGLE, C.L. CHANG, et R.C.T. LEE. « A new algorithm for genrating prime implicants ». *IEEE Transactions on Computers*, 19(4):304–310, 1970.

[Smith & Grant 1995] Barbara M. SMITH et Stuart A. GRANT. « Where the Exceptionnally Hard Problems Are ». Dans *Proceedings of the CP-95 Workshop on Studying and Solving Really Hard Problems*, pages 172–182, septembre 1995.

[Smullyan 1968] R. M. SMULLYAN. *First Order Logic*. Springer Verlag, New-York Inc, 1968.

[Tarjan 1972] Robert Endre TARJAN. « Depth First Search and Linear Graph Algorithms ». *SIAM J. Comput.*, 1:146–160, 1972.

[Tison 1967] P. TISON. « Generalized consensus theory and application to the minimization of boolean functions ». *IEEE Transactions on Electronic Computers*, 4:446–456, 1967.

[Tovey 1984] Craig A. TOVEY. « A Simplified NP-complete Satisfiability Problem ». *Discrete Applied Mathematics*, 8:85–89, 1984.

[Tseitin 1968] G.S. TSEITIN. « On the complexity of derivations in the propositional calculus ». Dans H.A.O. SLESENKO, éditeur, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[Turing & Girard 1991] Alan TURING et Jean-Yves GIRARD. *La machine de Turing*. Édition du Seuil, collection source du savoir, 1991.

[Van Gelder & Tsuji 1996] Allen VAN GELDER et Yumi K. TSUJI. « Satisfiability Testing with More Reasoning and Less Guessing ». Dans D.S. JOHNSON et M.A. TRICK, éditeurs, *Second DIMACS Challenge*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, pages 559–866, 1996.

[Yamasaki & Doshita 1983] S. YAMASAKI et S. DOSHITA. « The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in propositional logic ». *Journal of Information and Computation*, 59:1–12, 1983.

[Zabih & McAllester 1988] Ramin D. ZABIH et David A. MCALLESTER. « A rearrangement search strategy for determining propositional satisfiability ». Dans *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88)*, pages 155–160, 1988.

[Zhang & Stickel 1996] H. ZHANG et M.E. STICKEL. « An Efficient Algorithm for Unit Propagation ». Dans *Proceedings of Mathematics and Artificial Intelligence Symposium*, Fort Lauderdale (FL USA), janvier 1996.

[Zhang & Zhang 1996] J. ZHANG et H. ZHANG. « Combining Local Search and Backtracking Techniques for Constraint Satisfaction ». Dans *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 369–374, juillet 1996.

Index

– V –

– W –

Abstract

Knowledge representation and reasoning is a key issue in computer science and more particularly in artificial intelligence. In this respect, propositional logic is a representation formalism that is a good trade-off between the opposite computational efficiency and expressiveness criteria. However, unless $P = NP$, deduction in propositional logic is not polynomial in the worst case. In this respect, two correlated problems are addressed in this thesis: the satisfiability and the logical compilation of propositional knowledge bases.

First, new contributions about the computational aspects of the SAT problem (i.e. the satisfiability of a propositional formula in conjunctive normal form) are obtained, with respect to both logically complete and incomplete computational approaches to SAT. In particular, a new local search method based on a systematic use of a tabu list is proposed and fine-tuned. Surprising experimental results about the optimal length of the tabu list are exhibited with respect to hard random kSAT instances. A generalization of Oxusoff and Rauzy's model partition model is also obtained. New combinations of incomplete (local) search procedures and complete (systematic) search ones based on the Davis and Putnam's procedure that exploit the complementary properties of both approaches are investigated. Finally, a relation between the probability of generating a positive literal and the peak of difficulty of random instances is exhibited.

Then, a new approach to logical equivalence-preserving knowledge compilation is proposed. The key idea is to transform the initial formula into a tractable cover, i.e. a logically equivalent set of tractable formulas considered in a disjunctive way. It generalizes the standard notion of (prime) implicants cover. Three specific cases are investigated; namely, covers with respect to one theory, carver-based tractable covers and hyper-implicants covers. Both theoretical and experimental results show a substantial and time saving with respect to standard approaches.

All algorithms in this thesis have been implemented as a uniform multi-strategies platform and validated in an extensive manner with respect to standard benchmarks and hard random instances.

Keywords: Knowledge representation, propositional logic, SAT, logic compilation, Davis & Putnam procedure, local search, tabu search, mixed methods, random generation.

Résumé

La représentation des connaissances et les problèmes d'inférences associés restent à l'heure actuelle une problématique riche et centrale en Informatique et plus précisément en Intelligence Artificielle. Le formalisme considéré dans cette thèse est la logique propositionnelle qui permet d'allier puissance d'expression et efficacité. Il reste que la déduction en logique propositionnelle ne peut admettre de solutions à la fois générales et efficaces. Cette thèse traite de la satisfaisabilité et de la compilation de bases de connaissances propositionnelles, deux problèmes directement connectés à la problématique de la déduction.

Après une présentation générale de la thématique abordée (partie I), la seconde partie de cette thèse est dédiée à l'étude des méthodes de résolution du problème SAT. Nous proposons différents algorithmes préservant ou non la complétude logique. En premier lieu, nous présentons un nouvel algorithme de recherche locale (TSAT) basé sur une utilisation systématique d'une liste de réparations interdites (méthode tabou). Les algorithmes de recherche locale sont très souvent assujettis à un réglage pointilleux de leurs paramètres. Un résultat empirique surprenant permet de régler automatiquement la taille de la liste tabou en fonction du nombre de variables pour des instances kSAT aléatoires. En ce qui concerne les algorithmes complets pour SAT, notre contribution porte sur une généralisation du théorème de partition du modèle. Nous montrons comment une application restreinte de cette généralisation associée à la procédure de Davis & Putnam (DP) permet de diminuer la taille de l'arbre de recherche parcouru. La complémentarité des méthodes complètes et incomplètes conduit naturellement à l'idée de faire coopérer ces méthodes. Nous proposons plusieurs schémas de coopération et montrons que de telles combinaisons sont extrêmement performantes en pratique. Nous terminons cette seconde partie par la présentation de résultats expérimentaux obtenus sur le modèle standard de génération d'instances aléatoires. Les résultats obtenus mettent en évidence un lien surprenant entre la probabilité de « positiver » un littéral et le pic de difficulté des instances aléatoires.

La troisième partie de cette thèse est consacrée au problème de la compilation logique de bases de connaissances. Nous proposons une nouvelle technique de compilation logique préservant l'équivalence, basée sur un raisonnement modulo des théories. Cette technique repose sur l'idée de déterminer un ensemble de formules traitables (vu comme leur disjonction) sémantiquement équivalent à la formule initiale. Cet ensemble de formules est appelé couverture traitable. Ce type de couverture généralise la notion de couverture d'impliquants (premiers). Nous en proposons trois cas particuliers : les couvertures modulo une théorie, les couvertures de simplifications traitables et les couvertures d'hyper-impliquants. Les résultats théoriques et pratiques obtenus montrent que les approches à base de couvertures traitables améliorent considérablement les approches existantes (gain de temps et d'espace).

Une plate-forme logicielle regroupant l'ensemble des algorithmes décrits dans cette thèse a permis de valider expérimentalement les algorithmes utilisés sur de larges classes d'instances.

Mots-clés : Représentation des connaissances, logique propositionnelle, SAT, compilation logique, procédure de Davis & Putnam, recherche locale, recherche tabou, méthodes mixtes, génération aléatoire.