Informatique ubiquitaire : techniques de curage d'informations perverties

THÈSE

présentée et soutenue publiquement le 07 décembre 2018

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Yacine Izza

Composition du jury

Rapporteurs :	Chu-Min Li	Université de Picardie
	Odile Papini	Université d'Aix-Marseille
Examinateurs :	Philippe Besnard	IRIT CNRS, Toulouse
	Éric Grégoire	Université d'Artois, directeur de thèse
	Bertrand Mazure	Université d'Artois
	Jean-Marie Lagniez	Université d'Artois, co-encadrant de thèse

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

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

Ce travail de thèse a été réalisé grâce à un contrat doctoral octroyé par le Conseil Régional des Hauts de France.

Je dédie cette thèse
à ma famille.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Contexte et motivations

L'ubiquitaire et l'internet des objets ont pour cible l'interaction de composants intelligents autonomes. Le développement de ces techniques nécessite que les connaissances acquises et encapsulées par les différents objets et composants puissent conserver une grande fiabilité à tout instant. Ainsi pour assurer cette fiabilité, il est utile de mettre en place des techniques pratiques permettant de débusquer les informations contradictoires dans ces composants et objets qui parfois conduisent ces derniers à présenter des comportements inappropriés et à prendre des décisions problématiques lorsqu'ils sont dotés d'une capacité de raisonnement déductif.

Comment gérer les connaissances contradictoires a été depuis longtemps et demeure un sujet de recherche majeur en intelligence artificielle. Lorsqu'un raisonneur doit exploiter une base de connaissances en présence d'incohérences [1], il est impératif pour lui de surmonter ces incohérences. En particulier, dans le cadre propositionnel standard, il est possible pour un système de raisonnement d'inférer une conclusion et sa contradiction à partir de prémisses contradictoires. Dans ce contexte, le raisonneur peut par exemple se focaliser sur un sous-ensemble de la base des connaissances qui est maximalement cohérent et dériver des conséquences à partir de ce sous-ensemble. Il adopte alors une attitude *crédule* et accepte de prendre le risque d'inférer des conséquences qui peuvent être en conflit avec les informations d'un autre sous-ensemble de la base. Sinon, le raisonneur peut appliquer une politique *sceptique* en se focalisant seulement sur une partie des informations qui ne contiennent aucune contradiction ; il ne prend que ce qui est partagé par toutes les branches alternatives sous-jacentes des contradictions. De tels paradigmes de raisonnement ont beaucoup été étudiés en intelligence artificielle (voir par exemple dans (Reiter 1980), (Grégoire 1990), (Grégoire *et al.* 2014b) et (Makinson et Schlechta 1991)), la plupart des études ont porté principalement sur des modèles conceptuels et logiques, mais les aspects calculatoires sont le plus souvent circonscrits à l'étude des problèmes de complexité dans les pires des cas. En conséquence, les études théoriques sur les paradigmes de raisonnement pour gérer les connaissances contradictoires sont abondantes dans la littérature. En particulier, beaucoup d'approches ont été développées : logiques paraconsistantes (de Kleer 1986), logiques non monotones (Ginsberg 1987), révision de croyances (Fermé et Hansson 2011), fusion de connaissances (Konieczny et Grégoire 2006), etc. Ces approches manquent souvent d'implémentation en pratique et sont rarement appliquées aux problèmes réels. Une des raisons en est que les résultats décourageants montrent que de nombreuses tâches de raisonnement appartiennent au second niveau de la *hiérarchie polynomiale* (Chen et Toda 1995), rendant les approches inapplicables en pratique.

Dans cette thèse, on s'intéresse à une approche possible pour extraire des sous-bases d'informations cohérentes. Cette dernière est traitée d'un point de vue pratique dans un cadre clausal booléen, il s'agit donc de mettre en œuvre un système de filtrage d'incohérences dans des bases contradictoires. Nous adoptons le formalisme de la logique propositionnelle pour représenter des informations contradictoires et ainsi nous bénéficions des avancées des technologies SAT pour calculer des ensembles d'informations maximaux satisfaisables (MSSes pour *Maximal Satisfiable Subsets*) ou leurs complémentaires les

1. Dans cette thèse "incohérent", "inconsistant" et "contradictoire" sont considérés comme synonymes.

ensembles d'informations minimaux rectificatifs (MCSes pour *Minimal Correction Subsets*) dans la base d'informations contradictoire.

Notons que le problème soulevé dans cette thèse se produit aussi bien lorsque l'on souhaite réconcilier des agents ou simplement des sources d'informations qui sont mutuellement contradictoires. On peut alors chercher à fournir une base de connaissances *consensuelle* où chaque agent est prêt à laisser tomber sa base initiale et à adopter la nouvelle base du consensus. Le concept de consensus peut être interprété en dynamique des informations comme un opérateur de fusion des informations appartenant à différents agents ou différentes sources. Ces sources bien qu'individuellement cohérentes, contiennent la plupart du temps des informations mutuellement conflictuelles. Le consensus permet donc de définir un ensemble d'informations à partir de ces sources dont l'union est incohérente, mais aussi de faire adopter des informations qu'aucune des sources initiales prise isolément ne permettrait d'inférer.

Plan de la thèse

Cette thèse est composée de deux parties. La première est consacrée à l'état de l'art et la deuxième regroupe les contributions de cette thèse.

Le chapitre 1 est un rappel des éléments de base de la logique propositionnelle, des notions fondamentales de la théorie de la complexité, notamment la hiérarchie polynomiale et du formalisme SAT ainsi que les mécanismes que contiennent des solveurs SAT modernes, en particulier la résolution en SAT incrémental. Ces points que nous venons de citer constituent les préliminaires techniques nécessaires pour la compréhension de la suite du mémoire. Le chapitre 2 quant à lui est consacré au problème de recherche d'un ensemble maximal satisfaisable (MSS) dans le cadre clausal booléen. Un état de l'art concernant les meilleures approches connues pour le calcul de un ou de tous les MSSes est établi. Le chapitre 3 aborde le concept de consensus pour un groupe d'agents négociateurs, où différentes formes de consensus sont définies. Nous nous focaliserons sur la notion de consensus, proposée par (Grégoire *et al.* 2016c), dans le contexte de la logique propositionnelle en relation directe avec nos contributions dans cette thèse.

Les chapitres 4 et 5 concernent nos contributions au problème d'extraction d'ensembles maximaux cohérents. Nous présentons, dans le chapitre 4, notre méthode originale pour l'énumération exhaustive des MCSes, quand cette énumération est possible. Cette technique consiste à calculer récursivement des familles de MCSes grâce au processus de *rotation de modèle* de manière heuristique et efficace. Une comparaison expérimentale avec les meilleures techniques connues est réalisée. Une variante du problème de la recherche d'un MSS est étudiée d'un point de vue calculatoire dans le chapitre 5. Dans ce problème l'ensemble maximal à extraire doit être cohérent avec un éventail de contextes hypothétiques potentiellement mutuellement contradictoires. Une approche incrémentale est proposée et évaluée empiriquement.

Les chapitres 6 et 7 concernent nos contributions au problème de recherche de consensus dans un groupe d'agents ou simplement des sources d'informations mutuellement contradictoires. Le chapitre 6 expose notre méthode incrémentale proposée pour le calcul d'un consensus maximal par rapport à l'inclusion. Une évaluation expérimentale de cette méthode sur un large panel de benchmarks confirme son efficacité en pratique. Dans le chapitre 7, nous proposons et étudions la notion de consensus admissible dans le cadre de la logique propositionnelle. Ce nouveau concept est un raffinement de la définition de consensus présentée dans (Grégoire *et al.* 2016c).

Publications et communications issues de cette thèse

- Éric GRÉGOIRE, Yacine IZZA, Du ZHANG. On admissible consensuses. *International Journal on Artificial Intelligence Tools*, 27(1) : 1–20, 2018.

- Éric GRÉGOIRE, Yacine IZZA, Jean-Marie LANGIEZ. Boosting MCSes enumeration. *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 1309–1315. 2018.

- Éric GRÉGOIRE, Yacine IZZA, Jean-Marie LANGIEZ. On computing one max-subset inclusion consensus. *In Proceedings of the 29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'17)*, pages 838–845. IEEE Computer Society, 2017.

- Éric GRÉGOIRE, Yacine IZZA, Jean-Marie LANGIEZ. On the Extraction of One Maximal Information Subset that Does not Conflict with Multiple Contexts. *in Thirty AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages 3404–3410, AAAI Press, 2016.

- Éric GRÉGOIRE, Yacine IZZA, Jean-Marie LANGIEZ. Extraction d'un sous-ensemble maximal d'informations qui soit cohérent avec des contextes mutuellement contradictoires. *Dans Journées Francophones de Programmation par Contraintes*, *JFPC'16*, pages 19-26, 2016.

État de l’art

Chapitre

1

Logique propositionnelle et problème SAT

Sommaire

La vérification de la satisfaction d'un ensemble de clauses booléennes, appelée communément problème SAT, est central dans beaucoup de domaines de l'informatique, d'un point de vue calculatoire et théorique. En pratique, et plus particulièrement dans les applications liées au raisonnement automatique et à l'intelligence artificielle, l'objectif est de développer des procédures efficaces, capables de résoudre des instances de ce problème. En théorie de la complexité, SAT représente un point majeur, puisqu'il est le problème de référence de la classe de complexité *NP-complet*, défini par Cook (Cook 1971).

Dans ce chapitre, nous rappelons les éléments de base de la logique propositionnelle. Donc, nous allons définir la syntaxe et la sémantique de celle-ci, puis nous verrons ce qu'est une forme clausale d'une formule propositionnelle (section 1.1). Ensuite, nous présentons en 1.2 quelques notions de base sur la théorie de la complexité dans le pire cas, notamment sur les classes de la hiérarchie polynomiale. Enfin, dans la section 1.3, nous exposons le problème SAT et les mécanismes d'un solveur SAT moderne.

1.1 Logique propositionnelle

Une logique se définit d'abord de manière syntaxique, la sémantique étant définie par la suite sur ce langage. Ainsi, le langage des formules propositionnelles est formé à partir d'assertions (propositions) et reliées entre elles avec des connecteurs usuels comme le **et**, **ou** et **non**. La sémantique repose sur la notion d'interprétation. Elle permet d'extraire un sens des formules.

1.1.1 Syntaxe

Définition 1.1. (Atome)
Une variable propositionnelle, appelée également **atome**, est une variable booléenne qui prend une valeur *vrai* ou *faux*. On utilise également « 0 » pour *faux* et « 1 » pour *vrai*.

Définition 1.2. (Langage propositionnel)

Un langage $\mathcal{L}$ de la logique propositionnelle est construit sur :

- un ensemble fini $\mathcal{V}$ de variables propositionnelles (ou atomes) ;
- les connecteurs logiques $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$ représentant respectivement, la négation, la disjonction, la conjonction, l'implication matérielle et l'équivalence logique ;
- les constantes booléennes $\top$ (*vrai*) et $\bot$ (*faux*) ;
- les symboles parenthèses « (» et «) ».

Une suite finie de ses éléments de $\mathcal{L}$ est alors une expression.

L'alphabet ne suffit pas à la définition d'un langage. Il faut aussi définir les règles qui permettent de structurer ce dernier.

Définition 1.3. (Formule)
L'ensemble des **formules** propositionnelles est défini comme le plus petit ensemble d'expressions construit à partir d'un langage $\mathcal{L}$, tel que :

- $\bot$ et $\top$ sont des formules ;
- si Σ est une formule alors $\neg\Sigma$ est une formule ;
- $\Sigma \wedge \Delta$ est une formule si Σ et Δ sont des formules ;
- $\Sigma \vee \Delta$ est une formule si Σ et Δ sont des formules ;
- $\Sigma \Rightarrow \Delta$ est une formule si Σ et Δ sont des formules.

Une formule appartenant à cet ensemble est alors dite formule bien formée.

Notation 1.1. L'ensemble des variables apparaissant dans une formule bien formée Σ est noté $\mathcal{V}_\Sigma$.

Exemple 1.1. Soit $\{a, b, c, d\}$ un ensemble de variables propositionnelles. $((a \vee b) \wedge (a \vee c \vee \neg d))$ est une formule bien formé, tandis que $((a \vee b) \wedge (c\neg d))$ ne l'est pas.

Définition 1.4. (Littéral)
Un **littéral** est une variable propositionnelle ℓ ou sa négation $\neg\ell$ (notée également $\bar{\ell}$), appelée respectivement, **littéral positif** et **littéral négatif**. La négation de ℓ est appelé aussi complémentaire de ℓ.

Exemple 1.2. Soit a une variable, a et $\neg a$ sont des littéraux respectivement positif et négatif. Le complémentaire de $\neg a$ est $\neg\neg a$ ($\neg\neg a = a$).

Dans la section suivante nous parlons de la sémantique des formules bien formées, c'est-à-dire sous quelles conditions un énoncé est *vrai* ou *faux*.

1.1.2 Sémantique

Définition 1.5. (Interprétation)
Une **interprétation** μ d'une formule propositionnelle Σ est une application qui associe à chaque élément de l'ensemble des variables propositionnelles $\mathcal{V}_\Sigma$ une valeur booléenne :

$$\mu : \mathcal{V}_\Sigma \rightarrow \{\textit{vrai}, \textit{faux}\}$$

L'ensemble des interprétations pouvant être construit sur les variables de Σ a un cardinal de $2^{|\mathcal{V}_\Sigma|}$.

Remarque 1.1. On représente souvent une interprétation μ par l'ensemble des littéraux qu'elle associe à *vrai*. Par exemple, $\mu = \{a, b, \neg c\} = \{a = vrai, b = vrai, c = faux\}$.

Définition 1.6. (Interprétation des formules)
Une interpretation donnée μ peut être étendue à l'ensemble des formules par les règles suivantes :

- $\mu(\bot)$ = *faux* ;
- $\mu(\top)$ = *vrai* ;
- $\mu(\neg\Sigma)$ = *vrai* si et seulement si $\mu(\Sigma)$ = *faux* ;
- $\mu(\Sigma \vee \Delta)$ = *vrai* si et seulement si $\mu(\Sigma) = vrai$ ou $\mu(\Delta) = vrai$;
- $\mu(\Sigma \wedge \Delta)$ = *vrai* si et seulement si $\mu(\Sigma) = vrai$ et $\mu(\Delta) = vrai$;
- $\mu(\Sigma \Rightarrow \Delta)$ = *faux* si et seulement si $\mu(\Sigma) = vrai$ et $\mu(\Delta) = faux$;
- $\mu(\Sigma \Leftrightarrow \Delta)$ = *vrai* si et seulement si $\mu(\Sigma) = \mu(\Delta)$.

À partir d'une interprétation μ, nous pouvons déterminer la valeur de vérité de toute formule en utilisant la sémantique usuelles des opérateurs logiques.

Définition 1.7. Sémantique des connecteurs logiques usuels

		Négation	Disjonction	Conjonction	Implication matérielle	Équivalence logique	Disjonction exclusive
Σ	Δ	$\neg\Sigma$	$(\Sigma \vee \Delta)$	$(\Sigma \wedge \Delta)$	$(\Sigma \Rightarrow \Delta)$	$(\Sigma \Leftrightarrow \Delta)$	$(\Sigma \oplus \Delta)$
$vrai$	$vrai$	$faux$	$vrai$	$vrai$	$vrai$	$vrai$	$faux$
$vrai$	$faux$	$faux$	$vrai$	$faux$	$faux$	$faux$	$vrai$
$faux$	$vrai$	$vrai$	$vrai$	$faux$	$vrai$	$faux$	$vrai$
$faux$	$faux$	$vrai$	$faux$	$faux$	$vrai$	$vrai$	$faux$

TABLE 1.1 – Sémantique des connecteurs logiques usuels.

Définition 1.8. (Formules équivalentes) Deux formules propositionnelles Σ et Δ sont équivalentes et on note $\Sigma \equiv \Delta$, lorsque celles-ci prennent la même valeur de vérité pour toutes les interprétations. En d'autres termes, Σ et Δ sont équivalentes si et seulement si pour toute interprétation μ de Σ, nous avons $\mu(\Sigma) = \mu(\Delta)$.

Remarque 1.2. On peut déduire de la table de vérité 1.1 l'équivalence des formules suivantes :

- $(\Sigma \Rightarrow \Delta)$ est équivalent à $(\neg\Sigma \vee \Delta)$;
- $(\Sigma \Leftrightarrow \Delta)$ est équivalent à $((\Sigma \Rightarrow \Delta) \wedge (\Delta \Rightarrow \Sigma))$;
- $(\Sigma \oplus \Delta)$ est équivalent à $\neg(\Sigma \Leftrightarrow \Delta)$;
- $\neg(\Sigma \vee \Delta)$ est équivalent à $(\neg\Sigma \wedge \neg\Delta)$;
- $\neg(\Sigma \wedge \Delta)$ est équivalent à $(\neg\Sigma \vee \neg\Delta)$.

Une interprétation n'est pas forcément définie sur l'ensemble des variables propositionnelle de la formule. Dans ce cas, l'interprétation peut être partielle ou incomplète, ce qui se définit formellement de la manière suivante :

Définition 1.9. (Interprétation partielle, complète et incomplète)
Soit Σ une formule propositionnelle. On désigne par :

- **interprétation partielle** toute interprétation μ telle que : $|\mu| \leq |\mathcal{V}_\Sigma|$;
- **interprétation complète** toute interprétation μ telle que : $|\mu| = |\mathcal{V}_\Sigma|$;
- **interprétation incomplète** toute interprétation μ telle que : $|\mu| < |\mathcal{V}_\Sigma|$.

Remarque 1.3. Dans la suite de ce manuscrit, lorsqu'aucune information n'est apportée sur la nature de l'interprétation, celle-ci est considérée comme complète.

Définition 1.10. (Modèle)
Une interprétation μ est un **modèle** pour Σ (ou μ *satisfait* Σ) lorsque celle-ci rend Σ vraie, c'est-à-dire : $\mu(\Sigma) = 1$ ($\mu(\Sigma) = \textit{vrai} = \top$).

Définition 1.11. (Formule satisfiable, insatisfaisable et valide)
On dit qu'un formule Σ est :

- **satisfiable** (ou **cohérente**) (ou **consistante**) s'il existe au moins un modèle pour Σ ;
- **insatisfaisable** (ou **incohérente**) (ou **contradictoire**) (ou **inconsistante**) s'il n'existe aucun modèle pour Σ ;
- **valide** si toute interpretation de Σ est un modèle de Σ. On appelle également une telle formule une **tautologie**.

Définition 1.12. (Conséquence logique)
Soient Σ et Δ deux formules propositionnelles. On dit que Δ est une **conséquence logique** de Σ et on écrit $\Sigma \models \Delta$, si et seulement si tout modèle de Σ est un modèle de Δ.

Le théorème suivant permet d'énoncer un résultat fondamental en démonstration automatique (preuve par l'absurde).

Théorème 1.1. (Déduction)
Soient Σ et Δ deux formules propositionnelles, $\Sigma \models \Delta$ si et seulement si $\Sigma \wedge \neg\Delta$ est une formule insatisfaisable.

Définition 1.13. (Fermeture déductive)
Soit Σ une formule propositionnelle. La **fermeture déductive** de Σ est l'ensemble $Cn(\Sigma) = \{\phi \in \mathcal{L} \mid \Sigma \models \phi\}$ des conséquences logique de Σ.

Dans la section suivante nous nous intéressons aux différentes formes normales que peuvent posséder toute formule propositionnelle.

1.1.3 Formes normales

Définition 1.14. (Terme) Un **terme** (ou **monôme**) est une conjonction finie de littéraux, c'est-à-dire une formule de la forme $(\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n)$ où chaque ℓ_i est un littéral.

Définition 1.15. (Clause)
Une **clause** est une disjonction finie de littéraux, c'est-à-dire une formule de la forme $(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n)$ où chaque ℓ_i est un littéral. On dit qu'elle est **tautologique**, lorsque qu'elle contient un littéral ℓ et son complémentaire $\neg\ell$, sinon **fondamentale**.

Définition 1.16. (Clause unitaire, binaire, ternaire et n-aire)
On dit d'une clause composée de :

- un seul littéral, qu'elle est **unitaire** ;
- deux littéraux, qu'elle est **binaire** ;
- trois littéraux, qu'elle est **ternaire** ;
- $n > 3$ littéraux, qu'elle est n**-aire**.

Définition 1.17. (Clause positive, négative, mixte et de Horn)
Une clause est dite :

- **positive** si et seulement si tous ses littéraux sont positifs ;
- **négative** si et seulement si tous ses littéraux sont négatifs ;
- **mixte** si et seulement si elle est composé de littéraux positifs et négatifs ;
- de **Horn** si et seulement si elle a au plus un littéral positif.

Définition 1.18. (Forme normale négative (NNF))
Une formule propositionnelle est sous **forme normale négative** (NNF) si elle est exclusivement constituée de conjonction, de disjonctions et de littéraux.

Exemple 1.3. Soit l'ensemble des variables $\mathcal{V} = \{a, b, c\}$ et la formule $\Sigma = (\neg a \vee b) \wedge (a \wedge \neg c)$, alors Σ est une NNF.

Définition 1.19. (Forme normale conjonctive (CNF))
Une formule propositionnelle est sous **forme normale conjonctive** (CNF) si elle est constituée d'une conjonction de clauses.

Exemple 1.4. Soit l'ensemble des variables $\mathcal{V} = \{a, b, c\}$ et la formule $\Sigma = (\neg a \vee b) \wedge (a \vee b \vee c) \wedge \neg b$. $(\neg a \vee b)$, $(a \vee b \vee c)$ et $\neg b$ sont des clauses. Leur conjonction Σ est une CNF.

Remarque 1.4. Une interprétation μ d'une formule CNF Σ est un modèle de Σ si et seulement si elle satisfait toutes les clauses de Σ.

Définition 1.20. (Forme normale disjonctive (DNF))
Une formule propositionnelle est sous **forme normale disjonctive** (DNF) si c'est une disjonction de termes.

Exemple 1.5. Soit Σ une formule propositionnelle telle que $\Sigma = (\neg a \wedge b) \vee (\neg a \wedge b \wedge c) \vee \neg b$.
$(\neg a \wedge b)$ $(\neg a \wedge b \wedge c)$ et $\neg b$ sont des termes. Leur disjonction Σ est une DNF.

Remarque 1.5. De la même manière que les clauses et les termes, les formules CNF et DNF sont respectivement représentées comme des ensembles de clauses et de termes.

Propriété 1.2. Toute formule de la logique propositionnelle peut être écrite sous une forme normale. De plus, la formule à transformer et celle obtenue sont équisatisfaisables.

Exemple 1.6. Soit la formule propositionnelle $\Sigma = (a \rightarrow b) \wedge (\neg(a \rightarrow c) \vee d)$, Σ peut s'écrire de la manière suivante :

- $(\neg a \vee b) \wedge ((a \wedge \neg c) \vee d)$ sous forme NNF ;
- $(\neg a \vee b) \wedge (a \vee d) \wedge (\neg c \vee d)$ sous forme CNF ;
- $(a \wedge b \wedge \neg c) \vee (\neg a \wedge b \wedge d)$ sous forme DNF.

Toutefois, la transformation classique sous forme normale peut nécessiter une croissance exponentielle de la taille de l'ensemble obtenu. Cependant, une approche proposée par (Tseitin 1968) permet de transformer en temps et espace linéaire toute formule en une formule sous forme normale conjonctive équisatisfaisable.

Dans la section suivante, nous rappelons les notions de base de la théorie de la complexité et nous verrons ce qu'est la complexité d'un problème et quels sont les différents niveaux de complexité de la hiérarchie polynomiale.

1.2 Théorie de la complexité

Les problèmes étudiés en informatique sont très variés et certains d'entres eux sont faciles et d'autres difficiles à résoudre. Par exemple, le problème de tri est considéré comme un problème facile. Trier une liste de nombres entiers dans un ordre croissant ou décroissant, même avec un million d'éléments dans la liste, est un problème *traitable*. Par ailleurs, le problème d'ordonnancement, par exemple ordonner des tâches sous des contraintes de temps à respecter, est un problème *intraitable*. La difficulté d'un problème se mesure en quantités de ressources nécessaires dans le pire cas, en temps d'exécution ou en espace mémoire en fonction de la taille des données fournies en entrée, dont a besoin un algorithme pour le résoudre. La théorie de la complexité est le domaine qui étudie formellement la difficulté intrinsèque des problèmes et cette étude tend à les classifier en des niveaux de complexités différents exprimés en temps ou espace mémoire. Ainsi, une classe de complexité regroupe un ensemble de problèmes dont les ressources (temps et espace mémoire) sont similaires pour leur résolution.

De manière générale, on considère le temps d'exécution comme la ressource la plus significative pour la résolution d'un problème, l'espace mémoire lui étant fortement lié. Ainsi, dans cette section, nous parlerons des classes de complexité des problème en faisant référence au temps d'exécution. Pour une présentation plus complète sur ce thème, le lecteur pourra se référer utilement aux ouvrages (Garey et Johnson 1979, Papadimitriou 1994a, Perifel 2014).

1.2.1 Notion de « problème »

Dans la théorie de la complexité, un problème est constitué de deux éléments : une entrée et une question ou une tâche à réaliser sur l'entrée, telle que chaque entrée est une instance particulière du problème. Selon la question contenue dans le problème, on désigne dans quelle catégorie ce problème appartient. Ainsi, on distingue deux types :

- Les problèmes de décision : ce sont les problèmes dont le but est de décider de l'existence ou non d'une solution parmi les choix possibles dans l'espace de recherche. Par exemple, on peut considérer le problème de décision qui détermine si deux nombres entiers a et b admettent un diviseur commun.
- Les problèmes de fonction : ce sont ceux qui consistent à produire un élément solution grâce à une fonction de calcul. Donc, la sortie d'un problème de fonction est plus complexe que celui d'un problème de décision où la sortie est simplement la réponse oui ou non une solution est admise pour ce problème. Pour illustrer avec un exemple, on peut considérer le problème de tri d'une liste d'entiers dans un ordre croissant. Pour cet exemple, la sortie est la liste triée.

La théorie de la complexité étudie essentiellement (mais pas seulement) les problèmes de décision. Cela peut paraitre limitatif à première vue, mais il important de souligner qu'il est toujours possible de transformer n'importe quel problème de fonction en un problème de décision de difficulté équivalente.

Définition 1.21. (Problème de décision)
Un problème $\mathcal{P}$ est un problème de décision, lorsque les solutions qu'il peut admettre sont : « oui » et « non » (ou encore « *vrai* » et « *faux* », 0 et 1, . . .).

Définition 1.22. (Problème de décision complémentaire)
Soit le problème de décision $\mathcal{P}$. Le complémentaire de $\mathcal{P}$, noté co$\mathcal{P}$, est le problème dont les instances ayant la réponse "oui", sont celles qui ne le sont pas pour $\mathcal{P}$.

1.2.2 Complexité d'un problème

En théorie de la complexité sont étudiés les problèmes décidables, c'est-à-dire les problèmes qui admettent des algorithmes retournant une solution en un temps fini. Ceux qu'on désigne par indécidables sont les problèmes pour lesquels il n'y a pas d'algorithmes corrects pour les résoudre. Pour ceux là, il n'y a pas moyen d'évaluer leurs complexités. La complexité d'un problème est fixée par la complexité du meilleur algorithme qui le résout, meilleur dans le sens où il nécessite moins de ressources. L'estimation de la complexité d'un algorithme s'effectue par le calcul du nombre d'opérations élémentaires nécessaires à l'exécution de cet algorithme sur une instance du problème. La taille de l'instance du problème joue un rôle important. En effet, pour la plupart des problèmes, plus la taille de l'entrée est grande, plus il faudra du temps pour trouver la solution. Ainsi, le temps de calcul d'un algorithme est évalué en fonction de la taille de son entrée. La complexité d'un algorithme est donnée par $\mathcal{O}(f(n))$, une analyse asymptotique du nombre d'opérations qu'effectue l'algorithme, majoré par un facteur constant près $f(n)$ où n est la taille de la donnée en entrée.

Exemple 1.7. Soit $f(n)$ définie comme suit :

- $f(n) = 3n^2 - n + 5$, alors la complexité asymptotique est : $\mathcal{O}(n^2)$
- $f(n) = 3n \log n + 30n - 4$, alors la complexité asymptotique est : $\mathcal{O}(n. \log(n))$
- $f(n) = 2^n + 3n^2$, alors la complexité asymptotique est : $\mathcal{O}(2^n)$

Définition 1.23. (Complexité d'un algorithme)
Considérons l'algorithme $\mathcal{A}$. Lorsqu'une instance donnée en entrée de l'algorithme est de taille n, on note $\mathrm{g}_{\mathcal{A}}(n)$ le nombre d'opérations élémentaires à son exécution. Un algorithme a une complexité en temps $\mathcal{O}f(n)$ si et seulement si :

$$\exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \mathrm{g}_{\mathcal{A}}(n) \leq c \times f(n).$$

Exemple 1.8. Considérons l'algorithme classique qui effectue un tri interne de données par insertion sur n éléments et prenons comme opérations élémentaires l'affectation et la comparaison. Cet algorithme effectue, dans le pire des cas, $n^2/2$ affectations et $n^2/2$ comparaisons. Par conséquent, cet algorithme a une complexité en temps de $\mathcal{O}(n^2)$.

Définition 1.24. (Algorithme polynomial)
Soit un problème de décision $\mathcal{P}$ dont la taille de l'instance est n. Un algorithme qui résout $\mathcal{P}$ est dit polynomial si et seulement si il est de complexité $\mathcal{O}(n^k)$, avec k un entier naturel.

Définition 1.25. (Algorithme exponentiel)
Soit un problème de décision $\mathcal{P}$ dont la taille de l'instance est n. Un algorithme qui résout $\mathcal{P}$ est dit exponentiel si et seulement si il est de complexité $\mathcal{O}(k^n)$, avec $k > 1$ un entier naturel.

Définition 1.26. (Complexité d'un problème)
La complexité d'un problème est la complexité dans le pire cas du meilleur algorithme qui le résout.

Exemple 1.9. Reprenons l'exemple 1.8. L'algorithme de tri interne de données par insertion, dont la complexité en temps est en $\mathcal{O}(n^2)$, est donc en temps polynomial. Pourtant, la complexité du problème de tri interne de données est en temps « quasi-linéaire » $\mathcal{O}(n.\log(n))$ puisque l'algorithme de tri de données par fusion est en temps $\mathcal{O}(n.\log(n))$. Notons que le problème de tri interne de données est donc traitable.

1.2.3 Classes de complexité

La définition des différentes classes de complexité des problèmes se sert d'un modèle de calcul. La machine de Turing est le modèle le plus connu et plus utilisé pour l'analyse de complexité. Elle permet de représenter de manière abstraite une machine de calcul, comme un ordinateur. Le temps mis par une machine de Turing *M* sur une instance *x* est le nombre d'étapes de calcul de *M(x)* pour retourner une solution. Cela revient en quelques sort à mesurer le temps d'exécution de l'algorithme pour résoudre le problème sur l'entrée *x*, mais cette mesure est indépendante de la puissance de l'ordinateur faisant tourner l'algorithme : c'est pourquoi on parle d'étapes de calcul et non pas de secondes. Il existe plusieurs types de machine de Turing : machine déterministe, machine non-déterministe, machine à oracle, etc. Les ordinateurs actuels ne peuvent être modélisés que par des machines de Turing déterministes. La simulation d'une machine non-déterministe sur une machine déterministe est démontrée de complexité exponentielle. Toutefois, les machines de Turing non-déterministes constituent un outil essentiel pour définir des classes de complexité.

Dans la suite, nous présentons les classes de complexité en temps des problèmes de décision. Ces classes de complexité sont définies grâce à un modèle de la machine de Turing.

Définition 1.27. (Classe *P*)
La classe *P* est l'ensemble des problèmes de décision résolubles par une machine de Turing déterministe en un temps polynomial par rapport à la taille de la donnée en entrée.

Définition 1.28. (Classe *NP*)
La classe *NP* est l'ensemble des problèmes de décision résolubles par une machine de Turing non-déterministe en un temps polynomial par rapport à la taille de la donnée en entrée.

Remarque 1.6. Une machine de Turing déterministe est aussi une machine de Turing non déterministe (il suffit de considérer le déterminisme comme un non déterminisme particulier où il n'y a à chaque étape qu'un seul choix possible). Ainsi, on admet que $P \subseteq NP$. En revanche, la question « $NP \subseteq P$? » reste encore ouverte. D'où la conjecture suivante.

Conjecture 1.1. On conjecture que $P \neq NP$.

Définition 1.29. (Classe *coNP*)
La classe *coNP* est l'ensemble des problèmes de décision tels que leurs complémentaires appartiennent à la classe *NP*.

Conjecture 1.2. On conjecture que $NP \neq coNP$.

Définition 1.30. (Classe *DP*)
Un problème $\mathcal{P}$ appartient à la classe *DP* s'il peut être écrit comme $\mathcal{P} = \mathcal{P}_1 \cap \mathcal{P}_2$ avec $\mathcal{P}_1 \in NP$ et $\mathcal{P}_2 \in coNP$.

La notion de réduction permet de comparer deux problèmes $\mathcal{P}$ et $\mathcal{Q}$. On dira que le problème $\mathcal{Q}$ n'est pas plus difficile que $\mathcal{P}$, si la résolution de $\mathcal{P}$ permet la résolution de $\mathcal{Q}$. En d'autres termes, pour résoudre $\mathcal{Q}$ on se ramène à $\mathcal{P}$ et on dit que $\mathcal{Q}$ se réduit à $\mathcal{P}$. La réduction consiste à transformer une instance x du problème $\mathcal{Q}$ en une instance x' du problème $\mathcal{P}$ telle que $x \in \mathcal{Q}$ si et seulement si $x' \in \mathcal{P}$.

Définition 1.31. (Réduction polynomiale)
Une réduction en temps polynomial d'un problème $\mathcal{Q}$ à un problème $\mathcal{P}$ est une fonction f calculable en temps polynomial telle que :

$$\forall x \in \mathcal{Q} \Leftrightarrow f(x) \in \mathcal{P}$$

Si une telle fonction f existe, on dira qu'il y a réduction de $\mathcal{Q}$ à $\mathcal{P}$.

La réduction permet de comparer les problèmes entre eux et donc de parler des problèmes les plus difficiles d'une classe de complexité. Nous présentons pour cela les notions de difficulté et de complétude.

Définition 1.32. (Problème difficile/complet)
Soit $\mathcal{P}$ un problème décidable et $\mathcal{C}$ une classe de complexité.

- $\mathcal{P}$ est dit *$\mathcal{C}$-difficile* si tout problème de la classe $\mathcal{C}$ lui est réductible.
- $\mathcal{P}$ est dit *$\mathcal{C}$-complet* si il est *$\mathcal{C}$-difficile* et appartient à la classe $\mathcal{C}$.

Ainsi, un problème $\mathcal{P}$ est dit *NP-complet* s'il appartient à la classe *NP* et que tout problème de *NP* se réduit à $\mathcal{P}$. Historiquement, le premier problème dont la *NP-completude* a été prouvée est le problème de satisfaction de formules CNF (appelé SAT) (Cook 1971).

Plus haut, nous avons présenté les classes de complexité *P*, *NP* et *coNP*. La classe *P* regroupe les problèmes dits faciles (ou traitables), tandis que les classes *NP* et *coNP* contiennent les problèmes dits difficiles (ou intraitables). Toutefois, il existe des problèmes étant encore plus difficiles que ceux de la classe *NP*. Ces problèmes ne peuvent pas être résolus en un temps polynomial par une machine de Turing non-déterministe. Par conséquent, il est nécessaire de définir d'autres classes de complexité situées au-delà de *NP*. La machine de Turing avec oracle sert de modèle de calcul pour définir une hiérarchie de classes de complexité qui étend la notion de classes *P*, *NP* et *coNP*. Intuitivement, une machine de Turing à oracle est une machine de Turing pouvant appeler un sous-programme, que lui est capable de résoudre un problème de n'importe quelle complexité en un temps constant (d'où le nom d'*oracle*).

Définition 1.33. Classe $\mathcal{C}^{\mathcal{A}}$
Soit $\mathcal{C}$ une classe de complexité. $\mathcal{C}^{\mathcal{A}}$ est une classe de complexité des problèmes résolubles par une machine de Turing à oracle $\mathcal{A}$ en temps polynomial par rapport à la taille de l'entrée.

Définition 1.34. (Classes Δ_i^P, Σ_i^P et Π_i^P)
Les familles des classes de complexité Δ_i^P, Σ_i^P et Π_i^P (pout $i \geq 1$) sont définies inductivement comme suit :

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$;
- $\Delta_i^P = P^{\Sigma_{i-1}^P}$;
- $\Sigma_i^P = NP^{\Sigma_{i-1}^P}$;
- $\Pi_i^P = co\Sigma_i^P$.

Ainsi, $\Delta_1^P = P$, $\Sigma_1^P = NP$ et $\Pi_1^P = coNP$, etc.

Définition 1.35. (Hiérarchie polynomiale)
La hiérarchie polynomiale *PH* est l'union des Σ_i^P (pour $i \geq 0$) :

$$PH = \bigcup_{i \geq 0} \Sigma_i^P$$

La classe Σ_i^P est appelée i-ème niveau de la hiérarchie polynomiale.

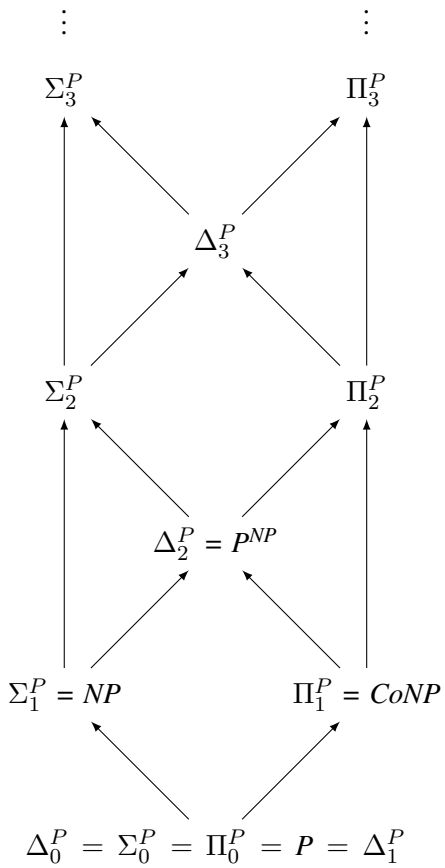

FIGURE 1.1 – Représentation graphique des premiers niveaux de la hiérarchie polynomiale.

La figure 1.1 est une représentation graphique des premiers niveaux de la hiérarchie polynomiale *PH*.

Comme nous pouvons le voir sur la figure 1.1, les problèmes sont classés de façon incrémentale, la classe d'un nouveau problème étant déduite d'un ancien problème. Il a toutefois été nécessaire de définir un « premier » problème *NP-complet* afin de classer tous les autres. Ce premier problème à avoir été démontré *NP-complet*, par (Cook 1971), est le problème de la satisfaisabilité propositionnelles (SAT).

1.3 Problème SAT

Le problème de satisfaisabilité booléenne (ou *Boolean Satisfiability Problem*), abrégé SAT, est sans doute l'un des problèmes de décision les plus importants et les plus étudiés en informatique. En effet, il est le premier problème de décision prouvé appartenant à la classe *NP-complet* (Cook 1971). Par conséquent, l'amélioration des performances des algorithmes dédiés à la résolution de ce problème est très importante pour beaucoup d'applications utilisant SAT. En effet, de plus en plus de problèmes équivalents à SAT, c'est-à-dire pouvant être réduits efficacement (en temps polynomial) à SAT, sont exprimés sous forme de logique propositionnelle dans le simple but de trouver une solution de manière efficace. Cette réduction permet souvent d'obtenir des performances pratiques supérieures aux méthodes classiques utilisées dans les différents domaines dont font partie chacun des problèmes. Ainsi, on va trouver des moteurs SAT dans la résolution de problèmes de *model checking* (Clarke *et al.* 2001), ordonnancement (Crawford et Baker 1994), ou encore la bio-informatique (Lynce et Marques-Silva 2006). Parfois,

le moteur SAT est caché – utilisé comme boîte noire – dans des applications de domaines variants de la planification (Kautz et Selman 1992; 1996), la vérification formelle (Stephan *et al.* 1996, Silva et Sakallah 2000, Velev et Bryant 2001), le diagnostic (Smith *et al.* 2004), jusqu'à la fouille de données (Davidson *et al.* 2010, Boudane *et al.* 2016), et même dans des solveurs aux pouvoirs d'expression plus étendus, comme dans les solveurs SMT (*Satisfiability Modulo Theory*).

La popularité de SAT est en partie due aussi à la simplicité de sa définition. De manière formelle, le problème SAT se définit comme suit :

Définition 1.36. (Problème SAT) Soit une formule CNF Σ. Le problème SAT est le problème de décision qui détermine si *oui* ou *non* Σ admet un modèle.

Il est souvent plus simple d'exprimer/modéliser un problème de décision donné, en logique propositionnelle. Puis, faire appel au solveur SAT qui lui répondra à la requête par *oui* (ou *non*) la formule reçue en entrée possède un modèle.

Comme mentionné précédemment, nous utilisons le mot SAT pour désigner le problème de décision de satisfaisabilité des formules propositionnelle sous forme CNF. Par abus de langage, on appelle chaque instance SAT, un problème SAT, et la procédure pour le test de satisfaisabilité un solveur SAT. Également, on emploie l'abréviation SAT et UNSAT pour dire que le problème est « satisfaisable » et « insatisfaisable », respectivement.

La suite de cette section est consacrée à la présentation des algorithmes de résolution du problème SAT. Notons tout de même, qu'il serait présomptueux de prétendre réaliser un inventaire complet de toutes les méthodes existantes dans la literature. Par conséquent, nous nous focalisons sur les approches les plus utilisées en pratiques, à savoir la procédure DPLL de Martin Davis, Hilary Putnam, George Logemann et Donald Loveland et l'approche CDCL qui est une extension de DPLL. Nous terminons ce tour d'horizon pour SAT par la présentation de la résolution SAT en mode incrémental, une autre manière d'appliquer le test de satisfaisabilité pour résoudre des problèmes au-delà de *NP*.

1.3.1 Résoudre le problème SAT

De nombreux algorithmes ont été proposés pour résoudre les instances de SAT. Les principes à la base de ces méthodes sont très variés : résolution (Robinson 1965, Galil 1977), réécriture (Boole 1854, Shannon 1940), énumération (Quine 1950, Davis *et al.* 1962, Jeroslow et Wang 1990), recherche locale (Selman *et al.* 1992), diagrammes binaires de décision (Akers 1978, Bryant 1992, Uribe et Stickel 1994), etc.

En pratique, les approches les plus utilisées sont : les algorithmes basés sur le principe de résolution et les algorithmes énumératifs reposant généralement sur la procédure DPLL (Davis *et al.* 1962). D'ailleurs, ces deux paradigmes sont à la base des solveurs SAT modernes. Ces solveurs, nommés CDCL (*Conflict Driving Clause Learning*) (Moskewicz *et al.* 2001a), sont en fait une combinaison fine de la procédure DPLL et d'une méthode basée sur le principe de résolution.

Résolution logique

La résolution de Robinson (Robinson 1965) est la règle d'inférence la plus utilisée pour le problème SAT. Ce mécanisme assez simple permet en effet de déduire une nouvelle clause à partir de deux autres clauses. Elle est notamment à la base de la procédure DP (Davis et Putnam 1960). Elle se définit ainsi.

Définition 1.37. (Règle de résolution)
Soient deux clauses α_1 et α_2 et x une variable booléenne, la règle de résolution est la règle d'inférence suivante :

$$(x \vee \alpha_1) \wedge (\neg x \vee \alpha_2) \vdash_{\mathcal{R}} \alpha_1 \vee \alpha_2$$

La clause $\alpha_1 \vee \alpha_2$ est appelée *clause résolvante* sur x des clauses $(x \vee \alpha_1)$ et $(\neg x \vee \alpha_2)$.

Propriété 1.3. Soient Σ une formule, α' la résolvante de $(x \vee \alpha_1)$ et $(\neg x \vee \alpha_2)$ deux clauses de Σ. Nous avons alors les deux propriétés suivantes :

- $(x \vee \alpha_1) \wedge (\neg x \vee \alpha_2) \models \alpha'$
- $\Sigma \equiv \Sigma \cup \{\alpha'\}$

Le règle de résolution est complète pour la réfutation, c'est-à-dire, si la CNF est insatisfaisable, alors on a la garantie qu'il existe une séquence finie de résolutions permettant de déduire la clause vide ($\alpha = \bot$).

Définition 1.38. (Dérivation par résolution et réfutation)
Une dérivation par résolution d'une clause α' à partir de Σ est une suite de clauses $\pi = [\alpha_1, \ldots, \alpha_k]$ telle que $\alpha_k = \alpha'$ et telle que pour tout $i \leq k$, soit (i) $\alpha_i \in \Sigma$; soit (ii) il existe α_m et α_n ($m < i$ et $n < i$) dans π tels que α_i soit la résolvante de α_m et α_n.

Toute dérivation par résolution d'une clause vide ($\alpha = \bot$) à partir de Σ est appelée *réfutation* ou *preuve*.

L'utilisation de la résolution de Robinson dans SAT consiste à appliquer la règle de résolution avec deux autres règles : la *subsumption* et la fusion. Le principe de la règle de *subsumption* est de retirer une clause s'il en existe une plus forte, dans le sens où les termes interdits par la première sont interdits par la deuxième. De manière formelle, la *subsumption* est définie comme suit :

Définition 1.39. (*Subsumption*)
Une clause $\alpha_1 = (a_1, \ldots, a_m)$ *subsume* la clause $\alpha_2 = (a_1, \ldots, a_m, b_1, \ldots, b_n)$ si α_2 contient tous les littéraux de α_1.

Exemple 1.10. Soient $\alpha_1 = (a \vee b)$ et $\alpha_1 = (a \vee b \vee c)$ deux clauses, la clause α_1 *subsume* la clause α_2.

Définition 1.40. (Règle de fusion)
La **règle de fusion** consiste à supprimer les occurrences redondantes d'un littéral dans une clause.

$$(a_1 \vee \cdots \vee a_m \vee \ell \vee b_1 \vee \cdots \vee b_n \vee \ell \vee c_1 \vee \cdots \vee c_p) \equiv (a_1 \vee \cdots \vee a_m \vee b_1 \vee \cdots \vee b_n \vee c_1 \vee \cdots \vee c_p)$$

Il est évident qu'un littéral redondant peut être retiré d'une clause, puisque cela ne changera pas l'ensemble (au sens mathématique) des littéraux de la clause, et ne changera donc pas l'ensemble des interprétations qui satisfont cette clause.

Le principe de Robinson suit un algorithme très simple une fois ces trois règles définies. En effet, la procédure consiste à appliquer l'une de ces trois règles un certain nombre de fois, en augmentant à chaque fois la formule CNF Σ de départ avec le résultat obtenu, le processus se termine soit par l'obtention de la clause vide, et donc Σ est insatisfaisable, soit par la saturation de l'ensemble des clauses (toute nouvelle résolvante est subsumée par une clause existante), auquel cas Σ est satisfaisable.

D'autres systèmes de preuve par résolution existent. Certains plus puissants comme la résolution étendue de (Tseitin 1983), d'autres plus faibles comme la résolution unitaire (Dowling et Gallier 1984).

Une preuve par *résolution unitaire* est une preuve par résolution où une des clauses à résoudre est une clause unitaire.

Malgré sa simplicité, le principe de résolution demeure relativement inefficace en pratique car le nombre de résolvantes à exploiter est souvent exponentiel, l'espace mémoire requis devient donc très important et le temps de calculs prohibitifs. Pour tenter de le réduire, il est courant de limiter le nombre de résolutions possibles à chaque étape, sans pour autant perdre la complétude pour la réfutation. Différentes restrictions basées sur la structure de la preuve par résolution ont été proposées. Elles sont généralement plus faciles à mettre en œuvre comme la résolution à la Davis Putnam (DP (Davis et Putnam 1960)).

Procédure DP

L'un des premiers algorithmes proposés pour résoudre le problème SAT est l'algorithme DP (Davis et Putnam 1960), introduit par Davis et Putnam en 1960. Bien qu'il soit initialement destiné à tester la validité des formules de logique du premier ordre [2], il est néanmoins plus connu comme une méthode de résolution du problème SAT. Il est, en effet, un raffinement de l'approche de Robinson dans la mesure où il génère de manière générale moins de clauses résolvantes. Le principe de la méthode DP est basé sur l'oubli successif des variables de la formule. Concrètement, l'algorithme peut être décrit de la manière suivante : étant donné une formule CNF Σ en entrée, l'algorithme commence par vérifier si la formule n'est pas trivialement satisfaite ($\Sigma = \emptyset$) ou falsifiée ($\bot \in \Sigma$). Si aucun de ces cas ne se présente, l'algorithme sélectionne une variable x de Σ, génère toute les résolvantes possibles sur cette variable, ce qui permet donc de supprimer toutes les clauses de Σ dans lesquelles la variable x apparaît. Le processus est itéré jusqu'à ce qu'une clause vide soit produite (Σ est UNSAT), ou la formule soit vide (Σ est SAT). À chaque itération, le sous-problème généré contient une variable en moins, mais avec un nombre potentiellement quadratique de clauses supplémentaires.

Toutefois, un inconvénient majeur se pose avec l'approche DP en raison du nombre de résolvantes à générer qui peut exploser, ce qui nécessite alors un besoin d'espace mémoire important. Or, dans les machines des années soixantes, la ressource mémoire était très limitée. Il y avait interêt alors d'échanger de la mémoire contre des temps d'executions plus lents. Ceci motiva alors les auteurs : Davis, Logemann et Loveland, à proposer une nouvelle version de DP, nommée algorithme DPLL (Davis *et al.* 1962). Cet algorithme contient les aspects principaux qu'on retrouve actuellement dans les algorithmes des solveurs SAT modernes.

Procédure DPLL

L'algorithme DPLL (Davis *et al.* 1962) est une amélioration de l'algorithme DP, mais à la difference de ce dernier, DPLL n'exploite pas explicitement la règle de résolution et il lui préfère la règle de division.

Il s'agit d'une procédure récursive décrivant un parcours en profondeur d'abord dans un arbre binaire où chaque noeud correspond à une interprétation partielle de la formule. Cette interprétation permet de simplifier la formule en tenant compte des affectations de variables déjà réalisées. Les feuilles de l'arbre correspondent soit à une contradiction (la sous-formule contient au moins une clause vide), soit à une solution (la sous-formule est vide).

La méthode DPLL, décrite dans l'algorithme 1.1, alterne entre une phase de branchement et une phase de retours-arrière chronologiques (ou *backtrack*). Le branchement consiste à choisir un littéral

2. La logique du premier ordre ou logique des prédicat se distingue de la logique propositionnelle par l'utilisation des quantificateurs (le quantificateur universel ($\forall$) et existentiel $\exists$) et des prédicats ou relations, sur ses variables.

ℓ (`pickBranchingVariable`) et à décomposer la formule Σ en deux sous-formules $\Sigma|_\ell$ et $\Sigma|_{\neg\ell}$. Ce principe est basé sur un constat simple : Σ est SAT si et seulement si $\Sigma|_\ell$ ou $\Sigma|_{\neg\ell}$ est SAT. Cette manière de faire permet de considérer les sous-formules de façon indépendante et également dans un ordre quelconque. Le *backtrack* consiste à remonter l'arbre de recherche jusqu'au nœud de décision précédent l'échec lorsqu'une clause vide est inférée, afin d'explorer une nouvelle branche de l'arbre.

Algorithme 1.1 : DPLL

Input : Σ une formule CNF
Output : SAT ou UNSAT
1 **if** $\Sigma = \{\emptyset\}$ **then return** SAT ;
2 **if** $\Sigma = \bot$ **then return** UNSAT ;
3 **if** $\exists \alpha \in \Sigma$ *tel que* $\alpha = (\ell)$ **or** $\exists$ *un littéral pur* ℓ *dans* Σ **then**
4 **return** DPLL $(\Sigma_{|\ell})$;
5 $\ell \leftarrow$ `pickBranchingVariable` (Σ) ;
6 **return** (DPLL $(\Sigma_{|\ell})$ **or** DPLL $(\Sigma_{|\neg\ell})$);

Naturellement, le processus de branchement et de *backtrack* suffisent pour une exploration complète de l'espace de recherche du problème. Toutefois, dans le but d'éviter le parcours d'un nombre important d'interprétations inutiles, de nouvelles règles sont à l'œuvre dans DPLL.

Propagation unitaire
La propagation unitaire, aussi connue sous le nom de propagation de contraintes booléennes, est l'équivalent sémantique de la résolution unitaire. Elle est l'un des procédés clés des solveurs SAT basés sur l'algorithme DPLL et qui est sans conteste la plus utilisée. Le principe de la propagation unitaire consiste à détecter des clauses dont tous les littéraux sont affectés à *faux* sauf un. Par conséquent, le littéral non affecté est *propagé* à *vrai*, puisqu'il est inutile de tester les deux valeurs booléennes de cette variable. Dans la procédure DPLL, la propagation s'effectue (ligne $3-4$) avant la phase de branchement sur une nouvelle variable libre. Les variables affectées grâce à la propagation unitaire sont appelées « variables propagées » et les variables sélectionnées pour le branchement sont appelées « variables de décision ». La définition formelle de la propagation unitaire est donnée ci-dessous.

Définition 1.41. (Propagation unitaire)
Soit Σ une formule, nous notons Σ^* la fermeture par propagation unitaire de Σ. Σ^* est définie récursivement comme suit :

- $\Sigma^* = \Sigma$ si $\nexists \alpha \in \Sigma$ telle que α est unitaire ;
- $\Sigma^* = \bot$ si Σ^* contient les deux clauses unitaires (ℓ) et $(\neg\ell)$;
- $\Sigma^* = (\Sigma|_\ell)^*$ tel que ℓ est un littéral apparaissant dans une clause unitaire de Σ.

Propriété 1.4. Une formule CNF Σ est consistante si et seulement si Σ^* est consistante.

Définition 1.42. (déduction par propagation unitaire)
Soient Σ une formule CNF et $x \in \mathcal{V}_\Sigma$. On dit que x est déduit par propagation unitaire de Σ, noté $\Sigma \models_* x$, si et seulement si $(\Sigma \wedge \neg x)^* = \bot$. Une clause α est déductible par propagation unitaire de Σ si et seulement si $\Sigma \wedge \bar{\alpha} \models_* \bot$, c'est-à-dire, $(\Sigma \wedge \bar{\alpha})^* = \bot$.

Définition 1.43. (littéral pur)
Un littéral ℓ est dit **littéral pur** pour un ensemble de clauses Σ si il apparaît uniquement positivement ou uniquement négativement dans Σ

Propriété 1.5. Soient Σ une formule CNF et ℓ un littéral pur de Σ, alors Σ et $\Sigma|_\ell$ sont équisatisfaisables .

Exemple 1.11. Soit la formule $\Sigma = (a \vee b) \wedge (\neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c)$. Le littéral $\neg c$ est pur, donc $\Sigma|_{\neg c} = (a \vee b)$.

Exemple 1.12. Soit la formule CNF suivante : $\Sigma = (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c \vee d) \wedge (b \vee c) \wedge (\neg a \vee b)$. La figure 1.2 illustre l'importance de la propagation unitaire et la simplification par les littéraux unitaires dans une procédure DPLL.

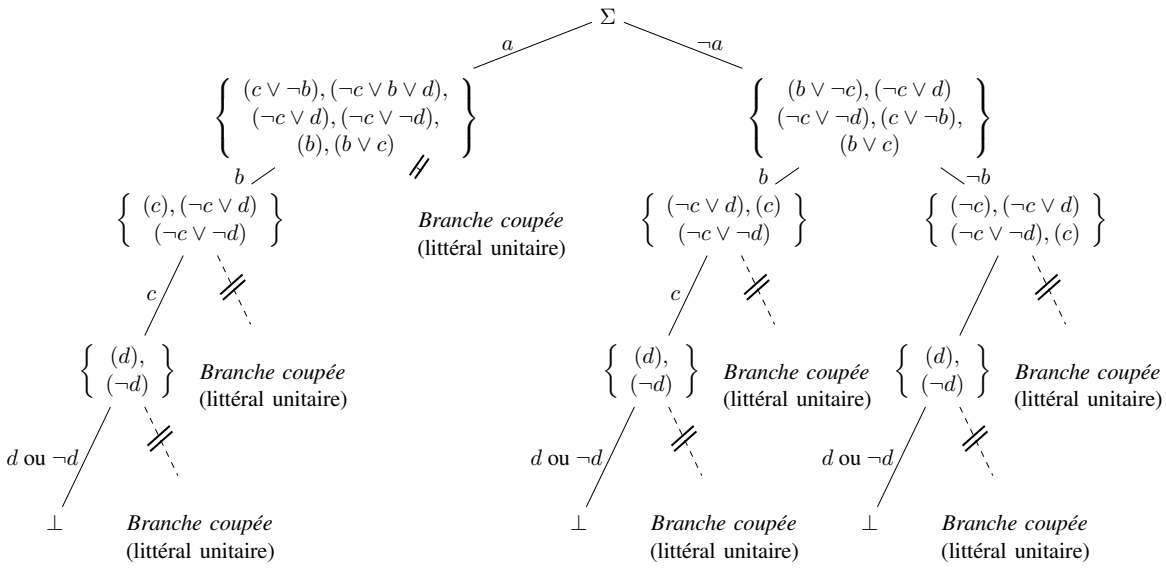

FIGURE 1.2 – Arbre de recherche construit par l'algorithme DPLL sur la formule CNF de l'exemple 1.12.

Définition 1.44. (Décision et niveau de décision)
Une **décision** est synonyme à l'affectation d'une variable ou d'un littéral via l'heuristique de choix de variable. Décider le littéral ℓ signifie affecter la variable ℓ à *vrai* et donc considérer $\Sigma|_\ell$.
Un littéral ℓ est au **niveau de décision** n si ce littéral est le n-ième littéral décidé ou si ce littéral a été propagé suite à une décision d'une variable au niveau n. Les littéraux propagés avant la première décision (qui existent seulement dans le cas où la formule contient des clauses unitaires) appartiennent au niveau de décision 0.

Définition 1.45. (Séquence décisions-propagations)
Soit une séquence de décisions $\langle x_1, \ldots, x_n \rangle$, et $y_{i,1}, \ldots, y_{i,p_i}$ les littéraux propagés après l'affectation du littéral x_i. La **séquence décisions-propagations** de $\langle x_1, \ldots, x_n \rangle$ est :

$$\{\langle \emptyset, y^0_{0,1}, \ldots, y^0_{0,p_0} \rangle, \langle (x^1_1), y^0_{1,1}, \ldots, y^1_{1,p_1} \rangle, \ldots, \langle (x^n_n), y^n_{n,1}, \ldots, y^n_{n,p_n} \rangle\},$$

où x^j_i désigne le littéral x_i et indique qu'il apparaît au niveau de décision j.

Comme nous l'avons souligné, la propagation unitaire est un processus fondamental de l'algorithme DPLL. Généralement, la propagation unitaire consomme 90% du temps CPU nécessaire à la résolution d'une instance. C'est en partie pour cette raison que l'implémentation de celle-ci a connu quelques évolutions dans les solveurs SAT modernes. Nous noterons la principale qui est les structures de données paresseuses (cf. 18).

Heuristiques de branchement
L'autre élément primordial est la fonction heuristique de choix de variable, et une grande partie des solveurs ne reposent que sur celle-ci. Ainsi, de nombreuses heuristiques de branchement ont été proposées pour tenter de limiter la taille de l'arbre de recherche.

Heuristique MOMS. L'heuristique MOMS (*Maximum Occurrences in Clauses of Minimum Size*) proposée par (Goldberg 1979), consiste à sélectionner la variable ayant le plus grand nombre d'occurrences dans les clauses les plus petites. L'intérêt de cette heuristique est de favoriser la propagation unitaire, afin d'obtenir le sous-problème le plus simple possible à résoudre.

Heuristique JW. L'heuristique JW (Jeroslow-Wang) (Jeroslow et Wang 1990) suit le même principe que l'heuristique MOMS, c'est-à-dire, choisir la variable apparaissant le plus dans les clauses courtes. Par ailleurs, dans JW, l'objectif est d'équilibrer entre les sous-arbres resultant du branchement sur la variable choisie. Par conséquent, la variable ayant une fréquence d'apparition équilibrée avec le littéral positif et négatif est privilégiée. Afin de satisfaire au mieux ces deux critères (fréquence d'apparitions et équilibre des signes), les auteurs de (Jeroslow et Wang 1990) proposent le calcul suivant :

$$\alpha \times \mathcal{S}_{JW}(x) \times \mathcal{S}_{JW}(\neg x) + \mathcal{S}_{JW}(x) + \mathcal{S}_{JW}(\neg x) + 1$$

où $\mathcal{S}_{JW}(x)$ (respectivement $\mathcal{S}_{JW}(\neg x)$) représente la mesure de représentativité du littéral x (respectivement $\neg x$) et où α est une constante fixée.

Heuristique BOHM. L'heuristique BOHM (Buro et Kleine-Büning 1992) a pour principe de satisfaire ou réduire les tailles des clauses courtes. Selon un ordre lexicographique, l'heuristique choisit la variable qui maximise le vecteur de poids $\mathcal{S}_{BOHM} = (s_1(x), s_2(x), \ldots, s_n(x))$ avec $s_i(x)$ calculé de la manière suivante :

$$s_i(x) = \alpha \times max(h_i(x), h_i(\neg x)) + \beta \times min(h_i(x), h_i(\neg x))$$

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

1.3.2 Solveur SAT moderne

Pendant très longtemps, les solveurs SAT de type DPLL qui se sont succédés, privilégiaient le calcul heuristique et devaient constamment mettre à jour de nombreux compteurs. À partir des années 90, des problèmes industriels, issus de domaines différents (comme exemple : model checking, planification, etc) font leurs entrées dans la communauté SAT. Ces problèmes sont caractérisés par le fait d'avoir des formules propositionnelles de grandes tailles, contenant des centaines de milliers de variables et des centaines de milliers de clauses. Face à ces formules de très grandes tailles, les solveurs n'étaient plus capables de faire face. Pour remédier à cela, de nouvelles structures de données plus optimisées ont été proposées, afin de réduire la charge des calculs des heuristiques. Plus encore, l'apprentissage de nouvelles clauses grâce à l'analyse précise des conflits rencontrés durant la recherche, entraina les moteurs SAT dans une nouvelle ère, celle des solveurs CDCL (*Conflict-Driven Clause Learning*), ou de manière abusive solveurs SAT modernes. Bien que les solveurs CDCL s'inspirent originellement de l'algorithme DPLL, il est évident que l'architecture de ces derniers a complètement évoluée. En effet, les mécanismes qui alimentent un solveur CDCL sont maintenant tournés vers l'apprentissage, offrant souvent l'image d'un algorithme de plus en plus éloigné de la recherche arborescente classique. À présent, nous allons décrire ces différentes techniques.

Apprentissage et retour-arrière non chronologique par analyse de conflits

Une des caractéristiques de l'algorithme CDCL est d'effectuer des retours-arrière de manière non chronologiques (*backjumping*) en prenant en compte les décisions qui sont à l'origine du conflit.

Nous avons vu dans la description de l'algorithme DPLL que, lorsqu'un un conflit est atteint, celui-ci remonte l'arbre de recherche au nœud de décision précédant l'échec. Le fait d'effectuer un retour-arrière chronologique peut dans certaines conditions être problématique. En effet, la cause de l'échec peut être due à un point de choix décidé plus haut dans l'arbre de recherche. Dans ce cas de figure, le solveur va explorer de manière répétitive les mêmes sous-arbres tant que l'origine de l'échec n'est pas considérée.

La technique d'analyse de conflit, proposée par (Silva et Sakallah 1996), remédie à ce problème en tenant compte de la source de l'échec. L'analyse de conflit permet d'effectuer un retour-arrière à un niveau $m < n$ sans nécessairement essayer toutes les possibilités entre le niveau du conflit n et le niveau m. Concrètement, ce processus consiste à extraire dans l'interprétation courante un sous-ensemble de littéraux (appelé ensemble conflit) qui est à l'origine de la dérivation de la clause vide. Ce sous-ensemble permet de déterminer la dernière décision dans la branche courante responsable du conflit. Un retour-arrière non chronologique est ensuite réalisé pour remettre en cause la valeur de vérité de cette variable.

Toutefois, malgré le fait d'effectuer un retour-arrière non chronologique il est possible qu'une même situation d'échec se répète dans le futur. Pour remédier à ce problème, les solveurs SAT modernes utilisent le concept d'apprentissage à partir des échecs cumulés lors de la recherche couplé avec le mécanisme du retour-arrière non chronologique (Silva et Sakallah 1996, Moskewicz *et al.* 2001a, Beame *et al.* 2004, Bayardo Jr. et Schrag 1997). La combinaison de ces techniques assure à la propagation unitaire d'être plus robuste à chaque apparition d'un nouveau conflit et permet au solveur de ne pas considérer la même situation d'échec plusieurs fois. L'approche mise en place est la suivante : pendant que le solveur effectue une recherche d'une solution, un graphe de dépendance est construit, ce graphe particulier est appelé « graphe d'implication ». Lorsqu'un conflit est atteint, une analyse de ce graphe est effectuée afin d'extraire un cube $(x_1, x_2, \ldots, x_n)$ signifiant que l'apparition du conflit est due à l'affectation conjointe des littéraux $x_1, x_2, \ldots, x_n$ à *vrai*. Cet ensemble de littéraux est ajouté sous forme de clause permettant ainsi de ne plus répéter le même échec lors de l'exploration d'un sous-arbre similaire. Cette clause, nommée « clause apprise » (*learnt clause*), est obtenue par la négation du cube construit par l'analyse du conflit $(x_1, x_2, \ldots, x_n)$. L'ajout d'une clause apprise, permet d'identifier et d'ajouter une clause impliquée par la formule à chaque fois qu'une contradiction est détectée. De plus, considérer cette clause pour la suite de la recherche permet à la propagation unitaire de découvrir de nouvelles implications lesquelles permettent d'éviter la répétition de la même situation d'échec.

Un graphe d'implication est un DAG (Graphe Acyclique Dirigé) représentant les dépendances entre clauses et les affectations obtenues par propagations des littéraux unitaires. La définition formelle d'un graphe d'implication est la suivante :

Définition 1.46. (Graphe d'implication)
Soient Σ une formule CNF et μ une interprétation partielle. Le graphe d'implication associé à Σ et μ est un DAG dans lequel tous les littéraux de μ sont représentés par un nœud et où une arête (x, y) étiquetée α apparaît pour chaque propagation unitaire par la clause α d'un littéral y lors de l'affectation ou la propagation d'un littéral x. Si deux littéraux contradictoires sont propagés, on ajoute un nœud $\perp$ et une arête entre chacun des littéraux contradictoires et le nouveau nœud $\perp$.

Exemple 1.13. Soit la formule CNF Σ suivante :

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

$\mu = \{\langle(\neg x_1^1), x_1^2, \neg x_1^3\rangle, \langle(x_2^4), \neg x_2^5, x_2^6\rangle, \langle(\neg x_3^7), \neg x_3^8, \neg x_3^9\rangle, \langle(\neg x_4^{10}), x_4^{10}, \neg x_4^{12}, \neg x_4^{13}, x_4^{14}, x_4^{15}, x_4^{16}\rangle\}$ est l'interprétation partielle obtenue par propagation de la séquence de décisions $\langle \neg x_1, x_4, \neg x_7, x_{10}\rangle$. Nous représentons en rouge (respectivement vert) les littéraux falsifiés de la formule (respectivement satisfaits) par l'interprétation μ. L'interprétation μ falsifie la formule (clause α_{13}). La figure 1.3 représente le graphe d'implication obtenu à partir de Σ et de l'interprétation μ.

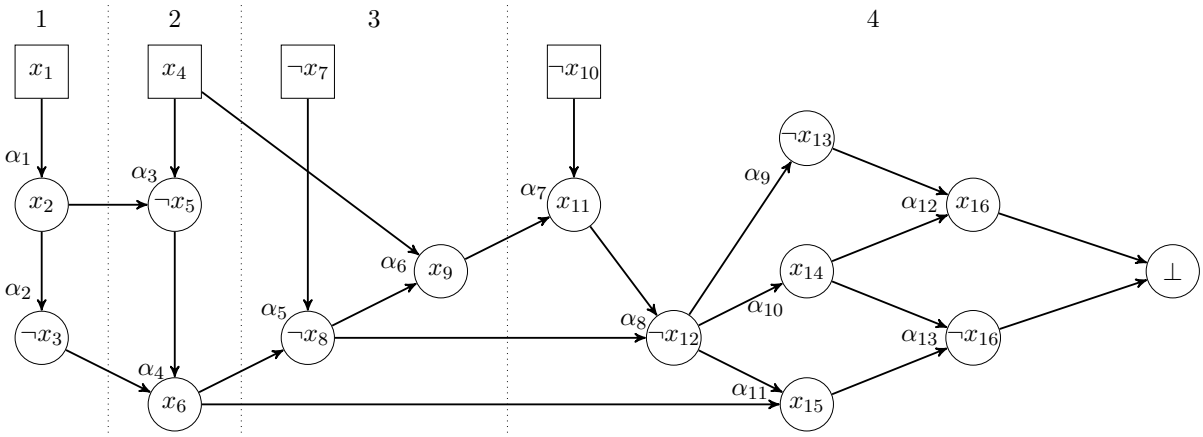

FIGURE 1.3 – Graphe d'implication obtenu à partir de la formule et l'interprétation partielle de l'exemple 1.13. Les nœuds de décision sont affichés en haut et sont représentés par des nœuds carrés. Les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'entre eux, la clause responsable de cette affectation est annotée. Le conflit se trouve sur la variable x_{16}.

Lorsque le graphe d'implication comporte un nœud $\bot$ symbolisant une situation de conflit, une analyse du graphe permet d'extraire les littéraux responsables du conflit. Généralement, cette analyse est basée sur la notion d'UIP (*Unique Implication Point*). Un UIP est un nœud du dernier niveau de décision qui « domine » le conflit. Formellement, nous avons :

Définition 1.47. (Point d'implication unique (UIP))
Soit $\mathcal{G}$ le graphe d'implication associé à une interprétation partielle conflictuelle. Soit $x \in \mathcal{G}$ le nœud associé au littéral choisi au dernier niveau de décision, et z le nœud de conflit (étiqueté par $\bot$ dans le graphe de la figure 1.3). Un nœud y est un nœud UIP si et seulement si tous les chemins de x vers z passent par le nœud y.

Remarque 1.7. Les UIP peuvent être ordonnés en fonction de leur distance avec le conflit. Le premier point d'implication unique (F-UIP pour « First Unique Implication Point ») est l'UIP le plus proche du conflit tandis que le dernier UIP (L-UIP « Last Unique Implication Point ») est le plus éloigné, c'est-à-dire le littéral de décision affecté au niveau conflit.

Exemple 1.14. Considérons le graphe $\mathcal{G}$ généré dans l'exemple 1.13. Les nœuds $\neg x_{12}$, x_{11} et $\neg x_{10}$ représentent respectivement le premier UIP, le second UIP et le dernier UIP.

L'intérêt d'utiliser les UIP dans l'analyse de conflit est de réduire la taille des clauses apprises. Actuellement, la plupart de solveurs SAT modernes s'appuient sur la notion de F-UIP afin de calculer la clause à ajouter dans la formule, nommée « clause assertive » (Moskewicz *et al.* 2001a). Cette clause est construite en appliquant la règle de résolution entre les clauses responsables du conflit (utilisé durant la propagation unitaire) en remontant celui-ci vers la variable de décision du dernier niveau jusqu'à obtenir une clause contenant un seul littéral du dernier niveau de décision (le UIP).

Exemple 1.15. Reprenons le graphe d'implication de l'exemple 1.3 dont le conflit porte sur la variable x_{16} et le littéral F-UIP est le littéral $\neg x_{12}$:

$$\begin{array}{rcl} \sigma_1 & = & \eta[x_{16}, \alpha_{12}, \alpha_{13}] = x_1 3^4 \vee \neg x_{14}^4 \vee \neg x_{15}^4 \\ \sigma_2 & = & \eta[x_{15}, \sigma_1, \alpha_{11}] = \neg x_6^2 \vee x_{12}^4 \vee x_{13}^4 \vee \neg x_{14}^4 \\ \sigma_3 & = & \eta[x_{14}, \sigma_2, \alpha_9] = \neg x_6^2 \vee x_{12}^4 \vee x_{13}^4 \\ \sigma_4 & = & \eta[x_{13}, \sigma_3, \alpha_{10}] = \neg x_6^2 \vee x_{12}^4 \end{array}$$

La clause assertive, correspondant au F-UIP ainsi déduite, est apprise par le solveur et permet non seulement d'éviter à l'avenir de répéter à nouveau cet échec, mais aussi d'effectuer un retour-arrière. En effet, la connaissance de cette clause falsifiée permet de calculer à quel niveau de décision le retour-arrière doit être effectué et ensuite propagé le littéral assertif (F-UIP).

Algorithme 1.2 : Solveur CDCL

Input : Σ une formule CNF
Output : SAT ou UNSAT

```
1  Δ ← ∅ ;                                              /* clauses apprises */
2  I_p ← 0 ;                                        /* interprétation partielle */
3  dl ← 0 ;                                              /* niveau de decision */
4  while true do
5      α = propagate(Σ ∪ Δ, I_p);
6      if α = null then
7          x ← pickBranchingVariable();
8          if toutes les variables sont affectées then return SAT ;
9          ℓ ← assignToPolarity(x);
10         dl ← dl + 1;
11         I_p ← I_p ∪ {ℓ^dl};
12         if réduire la base des clauses apprises then reduceDB(Δ);
13     else
14         (β, bl) ← analyzeConflict(Σ ∪ Δ, I_p, α);
15         if β = ⊥ then return UNSAT ;
16         Δ ← Δ ∪ {β};
17         backjump(Σ ∪ Δ, I_p, bl);
18         if redémarrage then restart(I_p, dl);
```

Actuellement, tous les solveurs SAT modernes partagent la même description (Algorithme 1.2) : une boucle principale alterne propagation unitaire et affectations de variables jusqu'à obtention d'un conflit. Chaque littéral choisi comme littéral de decision (ligne 7) est affecté suivant une certaine polarité (ligne 9) à un niveau décision donné (ligne 11). Si tous les littéraux sont affectés, alors μ_p est un modèle de Σ (ligne 8). À chaque fois qu'un conflit est atteint par propagation (lignes $14-15$), une clause β est calculée

par une procédure d'analyse de conflits et un niveau de *backjump* est calculé en fonction de la clause β (lignes $14-15$). À ce stade, il est possible de prouver l'incohérence (β est vide) de la formule (ligne 16). Si ce n'est pas le cas, un retour-arrière est effectué est le niveau de décision *dl* égal au niveau de *backjump bl* (ligne 18). Ensuite, certains solveurs CDCL forcent le redémarrage et dans ce cas, un retour-arrière est effectué au sommet de l'arbre de recherche (ligne 19). Enfin, la base de clauses apprises Δ peut être réduite lorsque celle-ci est considérée comme trop volumineuse (ligne 12).

Structures de données paresseuses (Watched Literals)

Cette technique fut introduite pour la première fois dans (Moskewicz *et al.* 2001a) et est devenue par la suite une partie intégrante des moteurs CDCL. L'idée est de mettre en place une structure de données spécifique afin de gérer efficacement la propagation unitaire. Intuitivement, étant donnée une clause α de n littéraux, la propagation unitaire est réalisée lorsque $n-1$ littéraux ont été falsifiés. Une idée simple et naïve pour effectuer cette opération consisterait à garder en mémoire le nombre de littéraux falsifiés pour α. Une fois que le compteur à atteint le nombre $n-1$, le littéral non affecté va être propagé. Or, ce qui est réellement important, c'est uniquement de savoir à quel moment le compteur est passé de $n-2$ à $n-1$, et dans ce cas appliquer la propagation. Pour cela, il suffit juste d'avoir des sentinelles (*watchs*) sur deux littéraux qui informent sur le statut (assignés ou non) de ces derniers. Si tous les deux littéraux ne sont pas affectés, alors aucun littéral ne peut être propagé. Par contre si un des littéraux est affecté et pas l'autre, alors deux situations se présentent :

- Tous les autres littéraux de la clauses sont à *faux*. Dans ce cas, le littéral surveillé qui reste est propagé ;
- Au moins un littéral ℓ dans la clause est toujours non assigné. Dans ce cas, la deuxième sentinelle est branchée sur ℓ.

Il est important de souligner que lorsqu'un *backtrack* a lieu, il est nullement nécessaire de bouger les *watchs* sur d'autres littéraux. Par conséquent, l'annulation d'une affectation de variable peut se faire instantanément.

Heuristique de choix de variable VSIDS

Les heuristiques classiques maintiennent une multitude de compteurs sur l'état courant de la formule réduite par les affectations courantes. Ces compteurs peuvent être simples, il en résulte pas moins que leur temps de calcul ou maintien a rapidement dominé le temps d'exécution des solveurs.

L'introduction des structures de données paresseuses (*Watched Literals*) dans les solveurs modernes a induit l'utilisation d'heuristiques non complexes et par conséquent moins gourmandes en terme de temps de calcul. VSIDS (*Variable State Independent Decaying Sum*) est un exemple typique des heuristiques paresseuses. Proposée pour la première fois par (Moskewicz *et al.* 2001b), elle calcule l'activité des variables indépendamment de l'affectation courante. Plus exactement, elle consiste à associer un compteur d'activité pour chaque littéral de la formule, ce compteur enregistre le nombre d'occurrences du littéral dans les clauses apprises. Au moment de choisir une variable à affecter, le littéral dont le compteur est plus élevé est sélectionné. Périodiquement, tous les compteurs sont divisés par une constante. Les versions les plus récentes de VSIDS diffèrent légèrement de la présentation originale. À l'instar du solveur MINISAT (Eén et Sörensson 2003a) qui utilise une version de VSIDS qui ne tient pas compte de la polarité de la variable.

Stratégie de redémarrages

La technique de redémarrage a été proposée initialement par (Gomes *et al.* 1997) (Gomes *et al.* 1998). Au fil du temps, son utilisation s'est imposée comme l'un des points clés de tout solveur SAT moderne. Le principe est de faire un retour-arrière jusqu'au sommet de l'arbre de recherche – niveau de décision zéro – et de relancer la recherche en l'orientant vers d'autres zones de l'espace de recherche. L'intérêt est d'éviter au solveur de passer trop de temps à chercher dans un sous-arbre qui a très peu de chance de fournir une solution. En effet, il arrive souvent que plus un solveur va passer de temps à chercher une solution, moins il a de chance de la trouver dans un temps restant court. Ceci peut s'expliquer par le fait que le solveur n'a pas branché sur les bonnes variables en haut de l'arbre et se perd donc dans un sous-arbre.

Réduction de la base de clauses apprises

L'analyse de conflits et l'apprentissage ont permis d'améliorer de manière significative l'algorithme CDCL (Silva et Sakallah 1996). Cependant, en ajoutant une clause après chaque conflit et puisque le nombre de conflits pendant une résolution peut être exponentiel dans le pire cas, le processus de propagation unitaire risque d'être ralenti. En effet, plus le nombre de clauses apprises est élevé, plus le processus de propagation est long. De ce fait, il est nécessaire de gérer l'accroissement de la base des clauses apprises au profit du processus de propagation unitaire (Eén et Sörensson 2003a). Pour ce faire, les solveurs CDCL appliquent tous une politique de réduction de la base des clauses apprises (`reduceDB`) en s'appuyant sur une fonction heuristique qui donne une estimation de la qualité des clauses. De cette manière, les clauses ayant les meilleurs scores de qualité sont les plus utiles et donc préservées dans la base. En revanche, pour les clauses ayant un score faible, elles sont supprimées dans la base selon un certain seuil fixé par rapport à la taille de la base. Par exemple, pour la plupart des approches, les clauses binaires sont gardées car elles permettent de contraindre fortement l'espace de recherche et par conséquent d'améliorer la propagation unitaires. En outre, les clauses « raisons » (à l'origine) des propagations sont systématiquement conservées au moment de l'appel à la fonction de réduction. En effet, pour rendre possible la construction du graphe d'implication et ainsi l'analyse de conflit, il n'est pas permis de supprimer une clause apprise intervenant dans la propagation unitaire.

Parmi les stratégies utilisées pour classer les clauses de la base apprises selon leur utilité, nous pouvons citer les plus populaires : la stratégie basée sur l'heuristique VSIDS appliquée dans (Eén et Sörensson 2003a). Celle-ci considère qu'une clause apparaissant peu dans la raison des conflits est inutile et qu'elle doit être supprimée. Une autre stratégie se base sur la mesure LBD (*Literal Block Distance*) proposée par (Audemard et Simon 2009). Cette mesure est statique et correspond au nombre de niveaux différents intervenant dans la génération de la clause apprise. La mesure PMS (*Progress Saving Measure*) a été proposée par (Audemard *et al.* 2011). L'heuristique basée sur PMS consiste à répartir l'ensemble des clauses apprises en trois sous-ensembles : actives, inactives et à supprimer.

1.3.3 Solveur SAT incrémental

Dans cette section, nous présentons un nouveau paradigme d'utilisation des solveurs SAT dans le but de résoudre des problèmes de complexité supérieure à *NP*. Pour la plupart de ces problèmes, la résolution consiste à appeler un solveur SAT sur plusieurs formules analogues. Ce type de résolution, appelé « résolution en SAT incrémental », est en passe de devenir l'état de l'art dans bien des domaines. Initialement intégré dès les premières versions du solveur MINISAT (Eén et Sörensson 2003a), ce mécanisme est utilisé dans diverses applications, à l'exemple de la vérification formelle bornée Eén et Sörensson

(2003b), Whittemore *et al.* (2001), Strichman (2001), l'extraction de sous-ensembles de clauses minimaux insatisfaisables (MUSes) ((Bailey et Stuckey 2005, Grégoire *et al.* 2007b, Liffiton et Sakallah 2008, Nadel 2010, Nöhrer *et al.* 2012, Previti et Marques-Silva 2013, Liffiton *et al.* 2016, Bacchus et Katsirelos 2015; 2016, Narodytska *et al.* 2018), etc) ou sous-ensembles de clauses maximaux satisfaisables (MSSes) ((Birnbaum et Lozinskii 2003, Rosa *et al.* 2010, Marques-Silva *et al.* 2013, Grégoire *et al.* 2014a, Bacchus *et al.* 2014, Mencía *et al.* 2015; 2016), etc), MAX-SAT (Fu et Malik 2006) ou encore la vérification inductive (Bradley 2012). Dans ce contexte, il ne s'agit pas de lancer un solveur SAT sur des formules CNF de très grandes tailles, mais potentiellement d'appeler le solveur un certain nombre de fois sur des instances proches les unes des autres avec des clauses ajoutées et/ou supprimées à chaque appel.

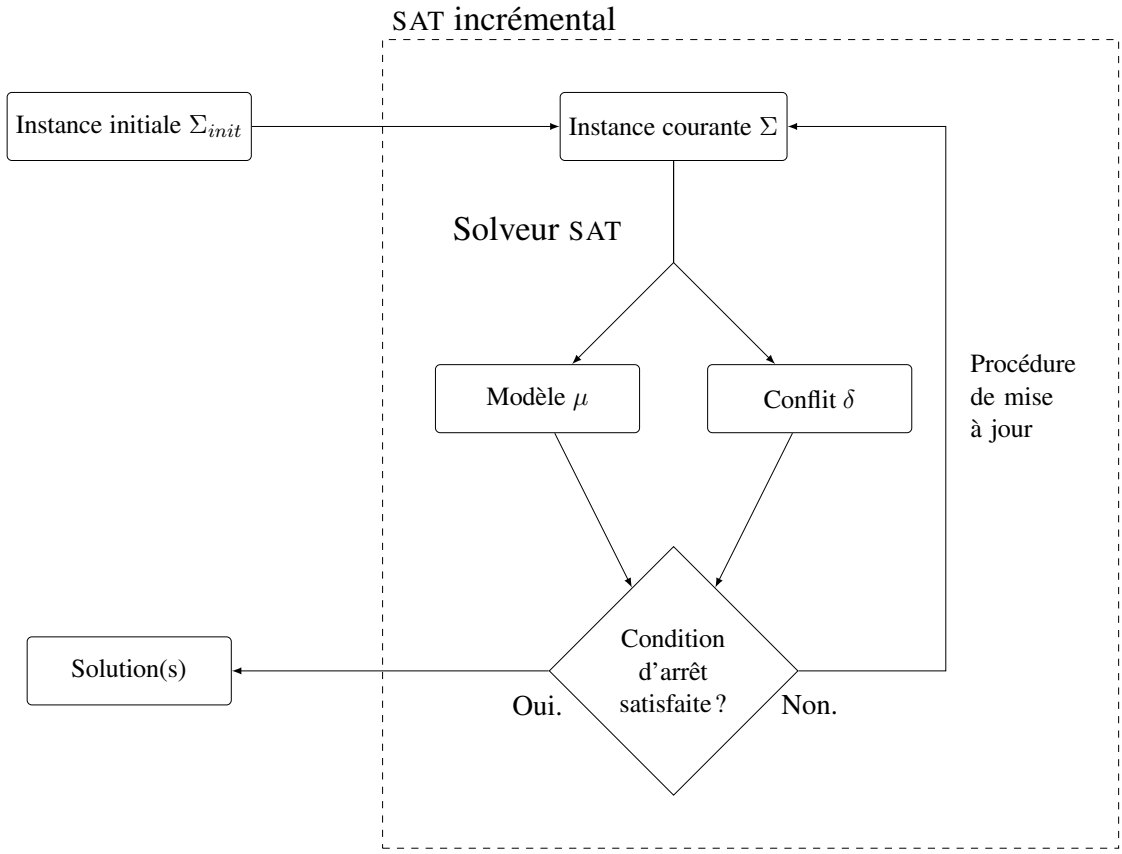

FIGURE 1.4 – Schéma général de l'architecture de la résolution incrémentale de SAT

La figure 1.4 résume le schéma général de l'architecture d'une approche basée sur le SAT incrémental. Comme on peut le voir sur la figure 1.4, la résolution en SAT incrémental est organisée comme suit :

1. Au départ, le moteur reçoit en entrée une formule CNF initiale Σ_{init} ;
2. Puis, le solveur est lancé sur la formule courante Σ. Une réponse (SAT ou UNSAT) est retournée à la fin (lorsque la formule courante Σ est SAT un modèle μ de Σ est fourni, sinon l'ensemble δ des littéraux responsables du conflit est délivré) ;
3. Si la condition d'arrêt est validée, alors une solution est retournée ;
4. Sinon mettre à jour la formule Σ en ajoutant et/ou supprimant un certain nombre de clauses.

L'idée de concevoir des solveurs SAT efficaces pour une exécution en mode incrémental a été motivée par la possibilité de pouvoir réutiliser les informations pouvant être collectées entre les appels successifs au solveur sur un ensemble de formules partageant des clauses entre elles. Initialement, une approche

de partage de clauses apprises est proposée (Strichman 2001, Whittemore *et al.* 2001), et consiste à choisir dans la base des clauses apprises celles qui semblent être pertinentes et à les propager entre les instances. En revanche, l'inconvénient de cette technique est de ne pas pouvoir exploiter directement les clauses apprises. Il est évident que l'ajout de nouvelles clauses à la formule, entre deux appels au solveur, ne remet pas en cause l'ensemble des clauses apprises. Par contre, supprimer des clauses dans la formule conduit à écarter les clauses apprises dérivées à partir de ces clauses supprimées. Pour pallier ce problème, (Eén et Sörensson 2003b) proposent une résolution incrémentale de SAT sous *hypothèses*. Dans cette approche, toutes les clauses apprises restent valides et sont des conséquences logiques pour toutes les instances. Cela permet au solveur de garder en cache toutes les clauses apprises générées à chaque appel. Concrètement, les hypothèses sont représentées par un ensemble $\mathcal{H}$ de littéraux qui sont choisis comme variables de décision et affectés à *vrai* au sommet de l'arbre de recherche (avant toute autre variable de décision). Ainsi, si durant la recherche une variable issue des hypothèses doit absolument changer de valeur de vérité dans l'interprétation courante pour satisfaire la formule, alors le problème est UNSAT suivant l'ensemble de littéraux $\mathcal{H}$.

SAT incrémental avec sélecteurs

Quand une hypothèse est utilisée pour *activer/désactiver* une clause, elle est appelée « sélecteur » pour cette clause. Le sélecteur étant une variable fraîche s_i ajoutée à une clause α_i, telle que la nouvelle clause générée $\alpha'_i = \neg s_i \vee \alpha_i$. La clause α'_i est une implication ($\alpha'_i = s_i \rightarrow \alpha_i$). Mettre en hypothèse le sélecteur s_i à *vrai*, implique la clause α_i. Inversement, assigner s_i à *faux* désactive α_i de la formule. Les sélecteurs n'apparaissant que négativement dans la formule, les clauses apprises obtenues au cours de la recherche gardent donc une empreinte (le sélecteur s_i associé) de toutes les clauses initiales de la formule utilisées pour les produire. Ainsi, en affectant à *faux* le sélecteur s_i, on désactive non seulement la clause associée mais également toutes les clauses apprises dérivées de celle-ci. Ceci est illustré dans l'exemple suivant.

Exemple 1.16. On considère une formule insatisfaisable CNF Σ où $\Sigma = \{\alpha_1 = (\neg a \vee b),\ \alpha_2 = (a),\ \alpha_3 = (\neg b),\ \alpha_4 = (\neg a \vee c),\ \alpha_5 = (\neg c)\}$, la formule augmentée Σ^S obtenue par l'ajout des variables sélecteurs est : $\Sigma^S = \{\alpha_1^s = (\neg a \vee b \vee \neg s_1),\ \alpha_2^s = (a \vee \neg s_2),\ \alpha_3^s = (\neg b \vee \neg s_3),\ \alpha_4^s = (\neg a \vee c \vee \neg s_4),\ \alpha_5^s = (\neg c \vee \neg s_5)\}$. Il est facile de voir que la formule Σ^S est satisfaisable (il suffit d'assigner toutes les variables sélecteurs $\{s_1, s_2, s_3, s_4, s_5\}$ à *faux* pour satisfaire toutes les clauses de Σ^S). Lorsqu'on choisit d'affecter comme hypothèses $s_1, s_2, \neg s_3, \neg s_4$ et $\neg s_5$ alors les clauses α_1 et α_2 sont activées pendant cet appel au solveur SAT, tandis que α_3, α_4 et α_5 sont désactivées. Si lors des précédents appels au solveur SAT on a appris la clause (les sélecteurs de ces deux clauses étaient alors *vrai*) issue de la résolution entre α_1^s et α_2^s et égale à $(b \vee \neg s_1 \vee \neg s_2)$ alors cette clause apprise ne sera activée que lorsque les hypothèses s_1 et s_2 seront vraies.

Calcul du conflit sous hypothèses

Dans le cas de la résolution en SAT incrémental, pour determiner les littéraux responsables de la clause vide (conflit), il suffit de considerer les nœuds racines (hypothèses) qui sont ancêtres de la clause vide. En effet, les nœuds racines qui ne sont pas ancêtres de la clause vide ne sont donc pas nécessaires pour construire la preuve d'insatisfaisabilité, et les supprimer ne remet pas en cause cette preuve. Ainsi, cette méthode nous assure l'obtention d'un sous-ensemble d'hypothèses qui est insatisfaisable.

Exemple 1.17. Considérons le formule CNF Σ^S de l'exemple 1.16 et supposons que nous avons comme hypothèse : $s_1, s_2, s_3, \neg s_4$ et $\neg s_5$, alors $\Sigma^S|_{s_1 s_2 s_3 \neg s_4 \neg s_5}$ est inconsistante comme le montre la figure 1.5

du graphe d'implication correspondant. Ainsi, $\delta = \{s_1, s_2, s_3\}$ est l'ensemble des hypothèses responsable du conflit et les clauses correspondant aux sélecteurs s_1, s_2 et s_3 forment un ensemble insatisfaisable, à savoir, l'ensemble $\{(\neg a \vee b),\ a,\ \neg b\}$.

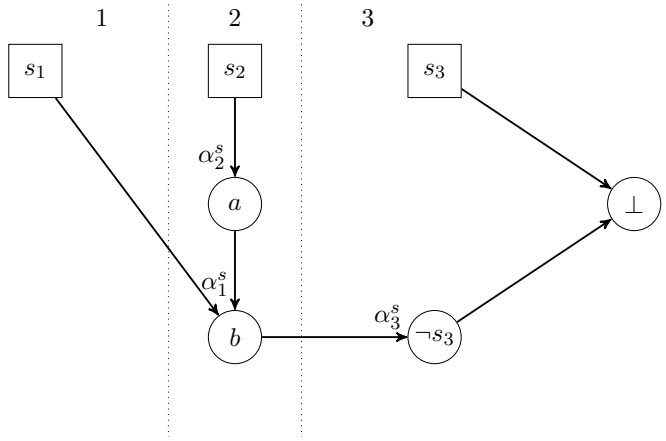

FIGURE 1.5 – Graphe d'implication obtenu à partir de la formule et les hypothèses de l'exemple 1.17.

Chapitre

2

Ensembles maximaux satisfaisables (MSSes)

Sommaire

Lorsqu'un ensemble de clauses Σ est incohérent, il est toujours possible d'extraire à partir de celui-ci un sous-ensemble de clauses qui soit satisfaisable. Un tel sous-ensemble peut être maximal [3] dans le sens où il n'est pas possible de l'étendre sans le rendre insatisfaisable et dans ce cas il est appelé MSS (*Maximal Satisfiable Subset*) de Σ. L'ensemble complémentaire d'un MSS dans Σ, noté CO-MSS ou plus souvent MCS (*Minimal Correction Subset*), désigne le sous-ensemble minimal de clauses à retirer de Σ pour obtenir un MSS. Ces deux notions de MSS et MCS jouent un role clé dans de nombreuses approches et techniques en intelligence artificielle, en allant des domaines de la représentation de connaissances, telle que la révision de croyance Fermé et Hansson (2011), Würbel *et al.* (2000), Benferhat *et al.* (2005), la fusion de connaissance Konieczny et Grégoire (2006), la théorie de l'argumentation (Besnard et Hunter 2008), raisonnement non monotone (McCarthy 1980, Ginsberg 1987), au domaine de diagnostic d'erreurs et débogage (Reiter 1987, Hamscher *et al.* 1992, Pereira *et al.* 1993, Felfernig *et al.* 2012, Marques-Silva *et al.* 2015) et la restauration de cohérence dans les bases de connaissances. Notons que le concept de MSS peut s'appliquer dans d'autres cadres que celui de la logique. Dans un système de contraintes par exemple et en particulier dans un problème de satisfaction de contraintes (CSP pour *Constraint Satisfaction Problem*), un MSS correspond à un ensemble maximal consistant de contraintes (voir, par exemple, (Rossi *et al.* 2006, Hemery *et al.* 2006)).

Dans le cadre de la logique propositionnelle, raisonner de manière crédule (Reiter 1980) sur une base d'informations contradictoire exprimée par une formule CNF peut se ramener à raisonner sur un MSS de la formule CNF et raisonner de manière sceptique peut se ramener à considérer l'intersection de tous les MSSes de la formule.

La recherche des sous-ensembles minimaux rectificatifs est très liée au problème de recherche des sous-ensembles minimaux insatisfaisables (MUSes pour *Minimal Unsatisfiable Subsets*). En effet, la notion de MUS et MCS sont très proches et il existe une dualité entre les deux ensembles MUSes et MCSes (Reiter 1987). On retrouve dans la littérature de nombreuses approches exploitant cette dualité pour calculer les MUSes et/ou les MCSes ((Reiter 1987, Grégoire *et al.* 2007b, Liffiton et Sakallah 2008, Nadel *et al.* 2014, Bacchus et Katsirelos 2015; 2016, Previti et Marques-Silva 2013, Liffiton *et al.* 2016, Narodytska *et al.* 2018), etc).

3. Maximal par rapport à l'inclusion ensembliste

Dans ce chapitre nous nous intéressons donc aux travaux réalisés ces dernières années au sujet du calcul des sous-ensembles maximaux satisfaisables dans le cadre SAT. Ce domaine a fait l'objet de nombreuses recherches cette dernière décennie et des avancées significatives ont été obtenues. Ce chapitre est organisé de la manière suivante : dans la prochaine section, sont définies de manière formelle les notions de MSS, MCS et MUS, ainsi que les résultats de complexité dans le pire cas quant à l'extraction d'un MSS. La section 2.2, quant à elle, est consacrée à la présentation des approches de calcul d'un MSS. Ensuite, dans section 2.3, sont exposées les approches pour l'énumération des MSSes. Finalement, la section 2.4 décrit la notion de sous-ensemble maximal satisfaisable avec des contextes multiples, noté AC-MSS, qui étend le concept de MSS.

2.1 Définitions et complexité

Étant donnée une formule CNF insatisfaible Σ. Un MSS (*Maximal Satisfiable Subset*) de Σ est un sous-ensemble satisfaisable des clauses de Σ où l'ajout d'une autre clause de Σ au MSS le rendrait insatisfaisable. Le complémentaire d'un MSS dans Σ est un MCS (*Minimal Correction Subset*), noté également CO-MSS, il représente un sous-ensemble minimal de clauses de Σ qu'il est nécessaire de retirer de Σ afin de rendre celle-ci satisfaisable. Les définitions formelles d'un MSS et MCS sont présentées ci-dessous.

Définition 2.1. (Sous-ensemble maximal satisfaisable (MSS))
Soient Σ une formule CNF et Γ un sous-ensemble de Σ ($\Gamma \subseteq \Sigma$). On dit que Γ est un **MSS** de Σ si et seulement si :

- Γ est satisfaisable ;
- $\forall \alpha \in \Sigma \backslash \Gamma, \Gamma \cup \{\alpha\}$ est insatisfaisable.

Définition 2.2. (Sous-ensemble minimal rectificatif (MCS))
Soient Σ une formule CNF et Γ un sous-ensemble de Σ ($\Gamma \subseteq \Sigma$). On dit que Γ est un **MCS** (ou CO-MSS) de Σ si et seulement si :

- $\Sigma \backslash \Gamma$ est satisfaisable ;
- $\forall \alpha \in \Sigma \backslash \Gamma, \Sigma \backslash (\Gamma \backslash \{\alpha\})$ est insatisfaisable.

La recherche d'un MSS (ou son complément MCS) dans une formule insatisfaisable peut servir dans nombreuses applications, mais toujours est il que ce problème s'avère d'une lourde complexité algorithmique. En effet, le problème de décision qui consiste à déterminer si un ensemble de clauses est un MSS d'une instance CNF a été prouvé comme appartenant à la classe de complexité *DP-complet* (Chen et Toda 1995).

Exemple 2.1. Soit la formule CNF insatisfaisable Σ formée par un ensemble de clauses :

$\alpha_1 = a \vee b$	$\alpha_4 = a \vee \neg b$	$\alpha_7 = \neg a \vee c$	$\alpha_{10} = \neg a \vee \neg c$
$\alpha_2 = \neg a \vee b$	$\alpha_5 = \neg b \vee c \vee d$	$\alpha_8 = \neg b \vee c \vee \neg d$	$\alpha_{11} = \neg b \vee c$
$\alpha_3 = \neg e \vee f$	$\alpha_6 = e \vee f$	$\alpha_9 = e \vee \neg f$	$\alpha_{12} = \neg e \vee \neg f$

$\Gamma = \{\alpha_1, \alpha_5, \alpha_6, \alpha_7, \alpha_8, \alpha_9, \alpha_{10}, \alpha_{11}, \alpha_{12}\}$ est un MSS de Σ, et le MCS correspondant est $\Sigma \setminus \Gamma = \{\alpha_2, \alpha_3, \alpha_4\}$. La formule Σ possède vingt quatre MSSes (et donc également vingt quatre MCSes).

Une formule CNF peut posséder plusieurs MSSes. En effet, elle peut même en contenir un nombre qui est exponentiel par rapport à la taille de la formule, comme le montre la propriété suivante :

Propriété 2.1. Dans le pire cas, une formule CNF insatisfaisable composée de n clauses possède en effet $C_n^{n/2}$ MSSes.

Preuve 2.1. Soit une formule CNF insatisfaisable Σ composée de n clauses. A priori, n'importe quel sous-ensemble de Σ peut être satisfaisable, et donc un de ces MSSes. Cependant, un MSS ne peut clairement pas en contenir un autre, étant donnée sa maximalité. Le problème de nombre de MSSes dans le pire cas se ramène donc au calcul du plus grand ensemble des parties de Σ, telle qu'aucune d'entre elle n'en contienne une aucune autre. Ceci revient donc au dénombrement du nombre maximal de sous-ensembles incomparables d'un ensemble à n éléments, qui est connu pour être $C_n^{n/2}$.

Remarque 2.1. Le nombre de MSSes d'une formule est égal au nombre de MCSes de cette dernière. Par conséquent, une formule CNF insatisfaisable composée de n clauses contient dans le pire cas $C_n^{n/2}$ MCSes.

Soulignons qu'un MSS peut être interprété comme étant une généralisation de la notion de MAX-SAT dans le problème SAT, puisque la maximalité dans le MSS est par rapport à l'inclusion ensembliste, tandis que pour le MAX-SAT elle est par rapport à la cardinalité. Ainsi, toute solution MAX-SAT est un MSS, mais l'inverse n'est pas toujours vrai (un MSS n'est pas nécessairement un MAX-SAT). Comme le préconise (Marques-Silva *et al.* 2013), des algorithmes spécifiques pour calculer un MSS peuvent s'avérer plus rapides que ceux dédiés à chercher une solution MAX-SAT. À cet égard, le calcul d'un MSS peut être utilisé pour fournir une solution approximative à MAX-SAT (voir par exemple (Marques-Silva *et al.* 2013, Bacchus *et al.* 2014, Mencía *et al.* 2015)).

Les notions de MSS et MCS peuvent être étendues aux concepts de Partial-MSS et Partial-MCS pour considérer le cas où l'instance Σ est constituée d'un couple $\langle \Sigma_1, \Sigma_2 \rangle$, où Σ_1 est l'ensemble des clauses dites « dures » de Σ et Σ_2 est l'ensemble des clauses « souples » de Σ : les clauses dures appartiennent à n'importe quel Partial-MSS de Σ ; il n'appartiennent donc à aucun Partial-MCS de Σ. Les definitions formelles d'un Partial-MSS et Partial-MCS sont données ci-dessous.

Définition 2.3. (Partial-MSS)
Soit la formule CNF $\Sigma = \langle \Sigma_1, \Sigma_2 \rangle$, où Σ_1 et Σ_2 représentent respectivement l'ensemble de clauses dures et souples de Σ. On dit que Γ est un Partial-MSS de $\langle \Sigma_1, \Sigma_2 \rangle$ si et seulement si :

- $\Sigma_1 \subset \Gamma \subseteq \Sigma_1 \cup \Sigma_2$;
- $\Gamma \cup \Sigma_1$ est satisfaisable ;
- $\forall \alpha \in \Sigma_2 \backslash \Gamma$, $\Gamma \cup \Sigma_1 \cup \{\alpha\}$ est insatisfaisable.

Définition 2.4. (Partial-MCS)
Soit la formule CNF $\Sigma = \langle \Sigma_1, \Sigma_2 \rangle$, où Σ_1 et Σ_2 représentent respectivement l'ensemble de clauses dures et souples de Σ. On dit que Γ est un Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$ si et seulement si :

- $\Gamma \subseteq \Sigma_2$
- $\Sigma_1 \cup \Sigma_2 \backslash \Gamma$ est satisfaisable ;
- $\forall \alpha \in \Sigma_2 \backslash \Gamma$, $\Sigma_1 \cup \Sigma_2 \backslash (\Gamma \backslash \{\alpha\})$ est insatisfaisable.

Il est important de noter que quand Σ_1 est insatisfaisable alors il n'existe aucun Partial-MSS pour Σ. De plus, chaque Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$ est un MCS de $\Sigma_1 \cup \Sigma_2$. Inversement, tout MCS de $\Sigma_1 \cup \Sigma_2$ qui ne satisfait pas Σ_1 n'est pas un Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$.

Exemple 2.2. Reprenons l'exemple 2.1 et considérons cette fois que les clauses de Σ sont réparties en deux sous-ensemble Σ_1 et Σ_2, où $\Sigma_1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\}$ constitue l'ensemble des clauses dures et $\Sigma_2 = \{\alpha_7, \alpha_8, \alpha_9, \alpha_{10}, \alpha_{11}, \alpha_{12}\}$ forme l'ensemble des clauses souples. Alors, $\Gamma = \{\alpha_{10}, \alpha_{11}, \alpha_{12}\}$ est un Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$, parmi d'autres.

Nous présentons à présent deux notions très connues dans l'analyse de l'inconsistance dans les formules CNF. Le CORE d'une formule CNF insatisfaisable Σ est un ensemble des clauses de Σ qui est insatisfaisable. Un MUS (*Minimal Unsatisfiable Subset*) de Σ est un sous-ensemble insatisfaisable des clauses de Σ qui est minimal, dans le sens où lorsqu'on réduit celui-ci il devient satisfaisable. Par conséquent, un MUS peut être vu comme étant un CORE ayant la propriété d'être minimal par rapport à l'inclusion ensembliste. De manière formelle, un CORE et MUS sont définis comme suit :

Définition 2.5. (CORE)
Soient Σ une formule CNF et Γ un sous-ensemble de Σ ($\Gamma \subseteq \Sigma$). On dit que Γ est un **CORE** de Σ si et seulement si Γ est insatisfaisable.

Définition 2.6. (Sous-ensemble minimale insatisfaisable (MUS))
Soient Σ une formule CNF et Γ un sous-ensemble de Σ ($\Gamma \subseteq \Sigma$). On dit que Γ est un **MUS** de Σ si et seulement si :

- Γ est insatisfaisable ;
- $\forall \alpha \in \Gamma, \Gamma \backslash \{\alpha\}$ est satisfaisable.

Exemple 2.3. Soit $\Sigma = \{\neg d \vee e,\ b \vee \neg c,\ \neg d,\ \neg a \vee b,\ a,\ a \vee \neg c \vee \neg e,\ \neg a \vee c \vee d,\ \neg b\}$ une formule CNF insatisfaisable composée de 8 clauses construites sur 5 variables. La figure 2.1 schématise, sous forme d'un diagramme de Venn, la formule et ces deux MUSes.

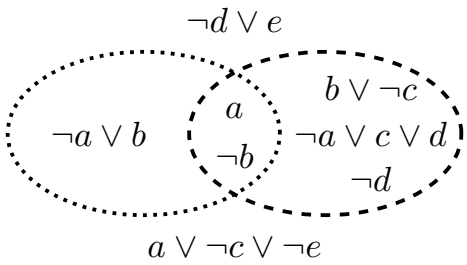

FIGURE 2.1 – Ensemble des MUSes d'une formule insatisfaisable.

Remarque 2.2. De manière similaire que pour les MSSes/MCSes, une formule CNF insatisfaisable contient au pire cas un nombre exponentiel de MUSes.

Remarque 2.3. Un CORE contient un ou plusieurs MUSes.

Exemple 2.4. Considérons la formule CNF Σ de l'exemple 2.3. Les ensembles $\{\neg a \vee b,\ a,\ \neg b\}$, $\{\neg d \vee e,\ \neg a \vee b,\ a,\ \neg b\}$ et $\{\neg a \vee b,\ a,\ \neg b,\ b \vee \neg c,\ \neg a \vee c \vee d,\ \neg d\}$ sont des CORES de Σ, parmi d'autres.

Notation 2.1. Soit Σ une formule CNF insatisfaisable, alors MSS$es(\Sigma)$, MCS$es(\Sigma)$ et MUS$es(\Sigma)$ désignent, respectivement, l'ensemble des MSSes, MCSes et MUSes de Σ.

Un concept très connu dans les approches de calcul d'un MCS (Grégoire *et al.* 2014a) et celles de calcul d'un MUS (Grégoire *et al.* 2007b, Previti et Marques-Silva 2013), est celui des clauses de transition. De manière intuitive, une clause de transition d'une formule inconsistante marque le passage de la formule de UNSAT à SAT en supprimant cette clause. La définition formelle d'une clause de transition est présentée ci-dessous.

Définition 2.7. (Clause de transition)
Soit Σ une formule CNF insatisfaisable. Une clause $\alpha \in \Sigma$ est dite **clause de transition** de Σ si et seulement si $\Sigma \backslash \{\alpha\}$ est satisfiable.

Remarque 2.4. Une clause de transition est un MCS de taille 1.

Exemple 2.5. Considérons à nouveau la CNF Σ de l'exemple 2.3 et considérons Σ' une sous-formule de Σ telle que $\Sigma' = \{\neg d \vee e,\ \neg a \vee b,\ a,\ \neg b\}$. Σ ne contient aucune clause de transition tandis que Σ' possède trois clauses de transition : $(\neg a \vee b)$, (a) et $(\neg b)$; chacune de ces clauses constitue un MCS de Σ'.

Dans la suite de ce chapitre nous dressons un panorama des approches de l'état de l'art pour le calcul de un et de tous les MCSes (section 2.2 et 2.3). Ces approches exploitent la richesse et la puissance des solveurs SAT modernes. Elles s'appuient sur la résolution incrémentale en SAT, où le solveur est appelé un certain nombre de fois, de manière incrémentale sur des sous-formules de la formule originale insatisfaisable. Par conséquent, le solveur SAT est utilisé comme un oracle *NP* qui vérifie la satisfaisablité de ces sous-formules.

La formule CNF originale Σ est formée d'un couple $\langle \Sigma_1 \cup \Sigma_2 \rangle$, où la partie à gauche Σ_1 représente l'ensemble des clauses dures de Σ et la partie droite Σ_2 désigne l'ensemble des clauses souples de Σ. Les clauses dures sont ajoutées directement dans le solveur SAT. Quant à la partie souple, chaque clause $\alpha_i \in \Sigma_2$ est augmenté par une nouvelle variable sélecteur s_i pour obtenir la nouvelle clause souple $(\alpha_i \vee \neg s_i)$ qui sera elle ajoutée au solveur SAT. L'utilisation des variables sélecteurs est la techniques standard adoptée pour activer et désactiver les clauses dans la résolution en SAT incrémental (Eén et Sörensson 2003b, Audemard *et al.* 2013). Les sélecteurs jouent alors le role d'hypothèses (*assumptions*) au début de chaque appel au solveur SAT pour activer/désactiver les clauses correspondantes. Concrètement, une clause $(\alpha_i \vee \neg s_i)$ est activée lorsque le littéral s_i est affecté à *vrai* et elle est désactivée quand s_i est mis à *faux*. L'ajout des sélecteurs rend la formule Σ satisfaisable (à condition que sa partie dure Σ_1 soit consistante). Lorsque tous les sélecteurs s_i sont à *vrai*, toutes les clause de Σ_2 sont activées et dans ce cas de figure la formule résultante est la CNF originale Σ. Les algorithmes de calcul de MCSes utilisent les sélecteurs comme *assumptions* pour pouvoir considérer des sous-formules de Σ et effectuer des appels SAT sur ces dernières. Dans la suite de chapitre, nous éviterons de faire référence explicitement aux variable sélecteurs, nous considérons plutôt une formule Σ avec tous les sélecteurs des clauses souples dans Σ_2 affectés à 1.

2.2 Approches d'extraction d'un MSS/MCS

Le calcul d'un MCS (ou MSS) d'une formule booléenne sous forme CNF est une tâche difficile. Comme nous l'avons souligné précédemment, vérifier si un sous-ensemble d'une formule CNF est un MCS est *DP-complet* (Chen et Toda 1995). D'un point de vue théorique, l'extraction d'un MCS est intraitable, néanmoins, en pratique un nombre important d'approches pour le calcul d'un MCS ont été proposées ces dernières années (Birnbaum et Lozinskii 2003, Rosa *et al.* 2010, Marques-Silva *et al.* 2013, Grégoire *et al.* 2014a, Bacchus *et al.* 2014, Mencía *et al.* 2015; 2016). Elles sont capables de résoudre en pratique des instances considérées comme non résolubles auparavant. En effet, les avancées qu'ont connues les technologies SAT avec l'apparition notamment des solveur SAT incrémentaux, ont permis d'élaborer de nouveaux algorithmes de plus en plus efficaces pour la détection de MCSes dans les formules CNF. Dans cette section, nous présentons les meilleures approches connues dans la littérature pour l'extraction d'un MCS dans une formule CNF. Nous parlerons également des améliorations introduites dans les différentes approches, elles permettent de réduire le nombre d'appels à l'oracle SAT et par conséquent d'augmenter les performances de ces algorithmes. Précisons qu'un avantage important s'offre dans ces améliorations est qu'elles peuvent se combiner entre elles et être intégrées dans les approches actuelles d'extraction d'un MCS.

Algorithme BLS

Basic Linear Search (BLS) est l'approche la plus basique pour l'extraction d'un MCS. Cette méthode consiste à effectuer une recherche linéaire des clauses de la formule qui sont mutuellement consistantes. L'algorithme 2.1 est le schéma qui décrit l'approche BLS. Une formule CNF Σ est donnée en entrée. L'algorithme débute par initialiser l'ensemble des clauses à tester U de Σ, et le MSS à construire Φ à $\emptyset$ (ligne 1). Ensuite, dans la boucle *while* (ligne $2-6$), tant que U n'est pas vide, l'algorithme retire une clause α de U et teste si $\Phi \cup \{\alpha\}$ est satisfaisable. Dans le cas vrai, α est ajouté dans le MSS Φ en construction. À la fin de la boucle, toutes des clauses de la formule ont été parcourues. Un MCS $\Sigma \backslash \Phi$, le complémentaire du MSS Φ, est fourni en sortie. Notons que l'algorithme BLS nécessite dans le pire cas $\mathcal{O}(n)$ appels aux solveur SAT.

Algorithme 2.1 : BLS (*Basic Linear Search*)

Input : une formule CNF Σ
Output : un MCS de Σ

1 $(U, \Phi) \leftarrow (\Sigma, \emptyset)$;
2 **while** $U \neq \emptyset$ **do**
3 $\quad \alpha \leftarrow$ **choose** $\alpha \in U$;
4 $\quad U \leftarrow U \backslash \{\alpha\}$;
5 $\quad (res,\ \mu) \leftarrow \text{SAT}(\Phi \cup \{\alpha\})$;
6 $\quad$ **if** $res = true$ **then** $\Phi \leftarrow \Phi \cup \{\alpha\}$;
7 **return** $\Sigma \backslash \Phi$;

Algorithme ELS

Enhanced Linear Search (ELS) est une amélioration de l'algorithme BLS, proposée dans (Marques-Silva *et al.* 2013). Dans cet algorithme, les auteurs proposent de greffer trois techniques pour la méthode par recherche linéaire, qui permettent d'un coté de réduire le nombre d'appels effectués au solveur SAT et de l'autre côté de simplifier la recherche dans solveur SAT. Nous verrons dans la suite que l'ensemble de ces techniques peut facilement être intégré, de manière totale ou partielle, dans les algorithme actuels pour l'extraction d'un MCS, comme c'est le cas pour l'algorithme CLD, FastDiag ou encore CMP. Dans la première technique, il s'agit de calculer dans la formule inconsistante un ensemble de CORES qui sont disjoints les un des autres et traiter chaque CORE séparément. Dans la deuxième, il s'agit d'exploiter les littéraux *backbone* (Kilby *et al.* 2005) afin d'améliorer les temps d'appel au solveur SAT. Enfin, la troisième technique consiste à exploiter le modèle donné par le solveur pour déplacer les nouvelles clauses satisfaites dans le MSS.

Calcul des CORES disjoints

Cette technique vise à décomposer la formule insatisfaisable Σ en sous-ensembles de clauses insatisfaisables U_i qui sont tous disjoints. La procédure DisjointCores (ligne 1. Algorithme 2.2) consiste à calculer de manière itérative les CORES en effectuant des appels au solveur SAT sur la formule Σ. En effet, tant que la formule Σ n'est pas satisfaite, le solveur renvoie un CORE responsable du conflit. Les clauses du CORE sont déplacées de Σ vers un ensemble U_i et un nouvel appel au solveur est appliqué sur Σ afin d'extraire un autre CORE. Ce processus itératif est répété jusqu'à ce que le solveur retourne la réponse SAT. Les clauses restantes de Σ sont consistantes et constituent ainsi l'ensemble initial du MSS Φ à élargir. À la fin, la procédure DisjointCores retourne une partition $\langle \Phi, \{U_1, \ldots, U_n\} \rangle$ de Σ.

Algorithme 2.2 : ELS (*Enhanced Linear Search*)

Input : une formule CNF Σ
Output : un MCS de Σ

1 $(\Phi, U) \leftarrow$ DisjointCores(Σ); // Φ est un MSS en construction et U est un ensemble des CORES disjoints $\{U_1, \ldots, U_n\}$.
2 $\Pi \leftarrow \emptyset$; // Π est un ensemble de littéraux backbone.
3 **foreach** $U_i \in U$ **do**
4 **while** $U_i \neq \emptyset$ **do**
5 $\alpha \leftarrow$ **choose** $\alpha \in U_i$;
6 $U_i \leftarrow U_i \backslash \{\alpha\}$;
7 $(res,\ \mu) \leftarrow \text{SAT}(\Phi \cup \Pi \cup \{\alpha\})$;
8 **if** $res = true$ **then**
9 $\Phi \leftarrow \Phi \cup \{\alpha\}$;
10 $\Phi \leftarrow \Phi \cup \{\beta \in U_i \mid \mu(\beta) = 1\}$;
11 $U_i \leftarrow \{\beta \in U_i \mid \mu(\beta) = 0\}$;
12 **else** $\Pi \leftarrow \Pi \cup \neg\alpha$;
13 **return** $\Sigma \backslash \Phi$;

L'utilisation des CORES disjoints permet, d'un côté, d'avoir une borne inférieure sur la taille de tout MCS et de l'autre côté, de construire un MCS en réparant localement chaque CORE.

Ajout des littéraux *backbone*
Cette technique est fondée sur la notion de *backbone* [4] (Kilby *et al.* 2005). Plus précisément, elle s'appuie sur l'observation suivante : lorsque la clause α est prouvée clause de transition du MSS Φ ($\Phi \cup \{\alpha\} \models \bot$), nous avons $\Phi \models \neg\alpha$, et pour chaque littéral $\ell_i \in \alpha$, $\Phi \models \neg\ell_i$. Ainsi, ℓ_i est un littéral *backbone* du MSS Φ. Dans l'algorithme 2.2, un ensemble Π capture les littéraux *backbones* (ligne 12). Ces littéraux sont ajoutés à la formule $\Phi \cup \{\alpha\}$ au moment de l'appel au solveur. La propagation des littéraux *backbones* permet de simplifier la recherche et ainsi réduire le temps de l'appel au solveur.

Exploitation des modèles retournés par le solveur
L'idée est initialement proposée dans (Nöhrer *et al.* 2012), puis ré-utilisée dans (Marques-Silva *et al.* 2013) pour le calcul d'un MCS. Le but ici est de déplacer un maximum de clauses vers le MSS en construction. Pour ce faire, il est demandé de chercher dans le modèle retourné par le solveur SAT lorsque la clause α est satisfiable avec Φ (lignes $7-8$. Algorithme 2.2) si il existe des littéraux de μ qui sont partagés avec des clauses de U_i. Dans ce cas, ces clauses sont donc satisfaites par le modèle courant μ et peuvent alors être déplacées vers le MSS courant Φ (lignes $10-11$. Algorithme 2.2). Cette technique offre clairement un avantage et permet d'éviter des appels supplémentaires au solveur SAT par le simple fait d'exploiter les modèles renvoyés en sortie par le solveur.

Algorithme CLD

CLD (*Computing one MCS with Clause D*) (Marques-Silva *et al.* 2013) s'inspire d'une propriété utilisée dans (Birnbaum et Lozinskii 2003) qui permet de considérer conjointement les clauses de la partie inconsistante au lieu d'évaluer les clauses une à une pour identifier celles qui sont clauses de transition à l'instar de l'approche BLS. Concrètement, l'idée de CLD consiste à appeler le solveur SAT

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

sur la formule $\Phi \cup \{D\}$ tels que Φ est le MSS à augmenter et D est la disjonction de tous littéraux appartenant à la partie inconsistante $\Sigma \backslash \Phi$. L'appel au solveur est itéré tant qu'il existe une interprétation qui satisfait $\Phi \cup \{D\}$. Ainsi, la clause D est satisfaite par l'interprétation et les littéraux de celle-ci qui sont à *vrai* représentent les clauses de $\Sigma \backslash \Phi$ qui sont consistantes avec Φ. Une mise à jour du MSS Φ en construction est effectué, les clauses de $\Sigma \backslash \Phi$ prouvées consistantes avec Φ sont déplacées vers celui-ci. La clause D est re-construite pour le prochain appel au solveur. Lorsque $\Phi \cup \{D\}$ n'admet plus de modèle ($\Phi \models \neg D$), l'algorithme finit son itération, plus aucune clause de $\Sigma \backslash \Phi$ ne peut être satisfaisable avec Φ. Par conséquent, le MSS final est atteint.

Dans l'algorithme 2.3 décrivant l'approche `CLD`, la clause D correspond à la disjonction des littéraux du CORE U_i en cours de traitement. En effet, pour `CLD`, les auteurs proposent d'utiliser la technique des CORES disjoints (voir la description dans la section 7), où chaque CORE U_i de la formule est analysé de manière indépendante. Il est intéressant de souligner que dans cette procédure, un appel au solveur SAT est économisé lorsque U_i contient une seule clause ($|U_i| = 1$). En effet, la clause restante dans U_i est forcément une clause de transition de Φ, car $\Phi \cup U_i \models \bot$. En plus de tirer avantage de la technique des CORES disjoints, `CLD` tire bénéfice aussi de la technique basée sur les littéraux *backbones* et celle des modèles calculés par le solveur.

Algorithme 2.3 : `CLD` (*Computing one* MCS *with clause D*)

Input : une formule CNF Σ
Output : un MCS de Σ

1 $(\Phi, U) \leftarrow$ `DisjointCores`(Σ); `// Φ est un MSS en construction et U est un ensemble des CORES disjoints` $\{U_1, \ldots, U_n\}$.
2 $\Pi \leftarrow \emptyset$; `// Π est un ensemble de littéraux backbone.`
3 **foreach** $U_i \in U$ **do**
4 **do**
5 $D \leftarrow (\bigvee_{\alpha \in U_i} \alpha)$;
6 $(res, \mu) \leftarrow \mathrm{SAT}(\Phi \cup \Pi \cup \{D\})$;
7 **if** $res = true$ **then**
8 $\Phi \leftarrow \Phi \cup \{\alpha \in U_i \mid \mu(\alpha) = 1\}$;
9 $U_i \leftarrow \{\alpha \in U_i \mid \mu(\alpha) = 0\}$;
10 **end**
11 **while** $((res = true)$ **and** $(|U_i| > 1))$;
12 $\Pi \leftarrow \Pi \cup (\bigcup_{\alpha \in U_i} \neg \alpha)$;
13 **end**
14 **return** $\Sigma \backslash \Phi$;

Algorithme `FastDiag`

`FastDiag` (Felfernig *et al.* 2012) est un algorithme dichotomique d'extraction d'un MCS, dédié au problème de diagnostic et débogage de programmes (Reiter 1987, de Kleer *et al.* 1992). Cette approche diviser-pour-régner s'inspire de la méthode QUICKXPLAIN conçue pour le calcul des MUSes (Junker 2004). L'organisation de la méthode `FastDiag` est décrite dans l'algorithme 2.4. Concrètement, il s'agit d'un algorithme récursif prenant en entrée trois arguments. Le premier argument Φ est l'ensemble de clauses à rendre consistant. Le second argument $\Psi \subset \Phi$ représente l'ensemble des clauses à retirer de Φ pour rendre celui-ci satisfaisable, autrement dit Ψ constitue le MCS à calculer. Initialement, le

triplet d'arguments que prend la procédure `FastDiag` correspond à $(\Sigma, \Sigma, \emptyset)$, où Σ est la formule CNF insatisfaisable fournie comme instance du problème. À chaque étape de la procédure, l'ensemble Ψ MCS en construction est divisé en deux sous-ensembles Ψ_1 et Ψ_2 (ligne 4). Un appel récursif est effectué sur chaque ensemble. Le résultat des deux appels (sur la partie gauche et partie droite) forme le MCS de la formule courante Φ.

Algorithme 2.4 : `FastDiag` (Φ, Ψ, Γ)

1 **if** $\Gamma \neq \emptyset$ **and** $\mathrm{SAT}(\Phi)$ **then return** $\emptyset$;
2 **if** $|\Psi| = 1$ **then return** Ψ ;
3 $k \leftarrow \frac{|\Psi|}{2}$;
4 $(\Psi_1,\ \Psi_2) = (\{\alpha_i \in \Psi \mid i = 0, \ldots, k\}, \{\alpha_i \in \Psi \mid i = k+1, \ldots, |\Psi|\})$;
5 $\Gamma_1 \leftarrow$ `FastDiag`$(\Phi \backslash \Psi_1, \Psi_2, \Psi_1)$;
6 $\Gamma_2 \leftarrow$ `FastDiag`$(\Phi \backslash \Gamma_1, \Psi_1, \Gamma_1)$;
7 **return** $\Gamma_1 \cup \Gamma_2$;

Notons que dans (Marques-Silva *et al.* 2013), une version améliorée de `FastDiag` est proposée. Dans cette nouvelle méthode, sont intégrées les techniques de littéraux *backbones*, exploitation des modèles et CORES disjoints décrites précédemment (voir 7).

Algorithme `CMP`

`CMP` (*Computational Method for Partitioning*) a été introduit par (Grégoire *et al.* 2014a). Dans cette approche, le paradigme de clause de transition est le concept principal. Cette approche consiste à partitionner progressivement une formule CNF inconsistante en un couple d'ensembles de clauses (MSS, CO-MSS) en effectuant des appels au solveur SAT pour détecter les clauses de transition et en utilisant certaines techniques optionnelles qui lui permettent de réduire le nombre d'appels au solveur et simplifier la résolution dans le solveur SAT. Bien qu'il existe un point commun entre `CMP` et `BLS` du fait qu'ils exploitent tous les deux le principe de clause de transition, ils restent différents sur la manière de procéder pour calculer ces clauses de transition. En effet, `BLS` est une approche constructive où à chaque itération de l'algorithme la partie MSS ou bien la partie MCS est augmentée par une nouvelle clause α selon le cas où α est une clause de transition ou non. À l'inverse de cela, `CMP` se présente comme une approche destructive. Elle commence le traitement itératif avec une formule inconsistante et à chaque itération une clause α de la formule est désactivée de manière temporaire, pour tenter de restaurer la cohérence de celle-ci. Si le retrait de α permet à la formule d'être saisfaisable alors α est une clause de transition et elle est définitivement retirée de la formule considérée et sauvegardée dans le MCS en construction. Une fois que la clause de transition est atteinte, les clauses préalablement désactivées sont ré-activées et le processus itératif est repris pour la recherche de nouvelles clauses de transition.

L'algorithme 2.5 synthétise l'organisation de l'approche `CMP`. Il prend en entrée une CNF Σ et retourne en sortie une partition de Σ en un couple (MSS, CO-MSS). Au départ, une partition approximative de Σ est réalisée grâce à la procédure `ApproximatePartition` qui effectue un appel au solveur SAT sur la formule Σ. Une interprétation complète de Σ est fournie par solveur. Les clauses de Σ satisfaites par l'interprétation sont copiées dans l'ensemble Φ qui constitue le MSS final en construction. Tandis que l'ensemble Ψ capture les clauses de Σ falsifiées par l'interprétation. L'ensemble Γ correspond à l'ensemble des clauses de transition, c'est-à-dire le MCS à construire. Initialement, Γ est vide (ligne 2). Un ensemble auxiliaire $U \subseteq \Psi$ est initialisé à Ψ (ligne 3). La suite du processus de l'algorithme consiste à répartir les clauses de Ψ entre le MSS Φ et le CO-MSS Γ tous deux en construction. Pour ce faire, l'algorithme cherche une clause $\alpha \in \Psi$ qui est une clause de transition de $\Phi \cup U$ telle que $\Phi \cup U$ est insatisfaisable et $\Phi \cup (U \backslash \{\alpha\})$ est satisfaisable. Lorsqu'une telle clause α est trouvée, elle est déplacée vers

Algorithme 2.5 : `CMP` (*Computational Method for Partitioning*)

input : une formule CNF Σ
output : une partition $(\text{MSS}, \text{CO-MSS})$ de Σ

```
1  (Φ, Ψ) ← ApproximatePartition(Σ); // Φ est un MSS en construction et
     Ψ est le complémentaire de Φ dans Σ.
2  Γ ← ∅;                            // Γ est un CO-MSS en construction.
3  U ← Ψ;
4  while Ψ ≠ ∅ do
5      extendSatPart(Φ, U, Γ, Ψ);
6      if Ψ ≠ ∅ then
7          α ← choose α ∈ U;
8          (res, μ, δ) ← SAT(Φ ∪ (U\{α}) ∪ ¬α);
9          if res = true then
10             Γ ← Γ ∪ {α} ;
11             Φ ← Φ ∪ {β ∈ (Ψ\{α}) | μ(β) = 1} ∪ ¬α ;
12             U ← Ψ ← {β ∈ (Ψ\{α}) | μ(β) = 0} ;
13         else
14             U ← U\{α} ;
15             exploiteCore(Φ, U, Ψ, δ, α);
16         end
17     end
18 end
19 return (Φ ∩ Σ, Γ) ;
```

le MCS Γ en construction (ligne 10). Les ensembles Φ, U et Ψ sont mis à jour en considérant le modèle μ retourné par le solveur. Lorsque α n'est pas une clause de transition, elle est exclue du sous-ensemble $U \subseteq \Psi$. Une nouvelle clause α à considérer est sélectionnée (ligne 7).

À présent, nous décrivons les différentes fonctionnalités optionnelles rajoutées dans `CMP` et laissées ouvertes lors de la description. La première fonctionnalité est implantée dans `extendSatPart`. L'idée de cette procédure est initialement appliquée dans (Birnbaum et Lozinskii 2003), puis dans (Marques-Silva *et al.* 2013) avec l'algorithme `CLD`. Elle consiste à vérifier la consistance de la formule $\Phi \cup \bigvee_{\alpha \in U} \alpha$. Dans le cas où $\Phi \cup \bigvee_{\alpha \in U} \alpha$ est UNSAT, cela implique que $\Phi \cup U$ l'est aussi. Par conséquent, les clauses de U sont déplacées vers Γ le MCS en construction. Dans le cas inverse, le sous-ensemble de clauses de U contenant des littéraux dans le modèle retourné par le solveur sont déplacées vers Φ le MSS en construction.

L'autre fonctionnalité utilisée dans `CMP` concerne l'exploitation des modèles (Nöhrer *et al.* 2012, Marques-Silva *et al.* 2013) retournés par le solveur SAT et l'intégration des littéraux *backbones* (Marques-Silva *et al.* 2013) (lignes $11 - 12$).

Une nouvelle fonctionnalité a été introduite dans `CMP`. Elle consiste à rajouter la négation de la clause α dans la formule sur laquelle est appelé le solveur pour tester si α est oui ou non une clause de transition. La propagation des clauses unitaires obtenues par $\neg\alpha$ permet d'accélérer le test de satisfaisabilité.

Enfin, la dernière amélioration proposée dans l'approche `CMP` est implémentée dans la procédure `exploiteCore`. L'idée consiste à exploiter le CORE δ retourné par le solveur SAT lorsque $\Phi \cup (U \backslash \{\alpha\})$ est vérifiée UNSAT. Le CORE indique quelles sont les clauses qui sont à l'origine du conflit (inconsistance). Ainsi, en analysant δ dans `exploiteCore`, celle-ci s'occupe de filtrer les clauses non impli-

quées dans conflit dans le sous-ensemble $U \subseteq \Psi$ où une clause de transition est pistée.

Pour conclure, soulignons que l'algorithme CMP nécessite, au pire cas, $\mathcal{O}(n^2)$ appels au solveur SAT, où n est le nombre de clauses dans la formule Σ. Bien que nous avons à côté l'approche BLS qui demande $\mathcal{O}(n)$ appels au solveur SAT en pratique CMP offre de meilleures performances.

Algorithme LBX

LBX (*Literal Based eXtractor*) est une approche introduite par (Mencía *et al.* 2015) qui part de l'idée de considérer les littéraux de la formule inconsistante et non pas ses clauses. Cette idée est inspirée de la correspondance qui existe entre les clauses du MCS et les littéraux *backbones* (Marques-Silva *et al.* 2013) du MSS.

Propriété 2.2. Soient Σ une formule CNF, Φ un MSS de Σ et une clause $\alpha \in \Sigma$. Alors, α est une clause du CO-MSS ($\Sigma \backslash \Phi$) si et seulement si pour tout littéral $\ell \in \alpha$, nous avons que $\Phi \models \neg\ell$, c'est-à-dire, ℓ est un littéral *backbone* de Φ.

En partant de la propriété 2.2, les auteurs de (Mencía *et al.* 2015) proposent un algorithme consistant à detecter itérativement les littéraux *backbones* de la partie MSS.

Algorithme 2.6 : LBX (*Literal Based eXtractor*)

input : une formule CNF Σ
output : un MCS de Σ

```
1  (Φ, U) ← approximatePartition(Σ);
2  L ← literals(U);                    // L est un ensemble de littéraux.
3  Π ← ∅;                              // Π est un ensemble de littéraux backbones.
4  while L ≠ ∅ do
5  |  ℓ ← RemoveLiteral(L);
6  |  (res, μ) ← SAT(Φ ∪ Π ∪ {ℓ});
7  |  if res = true then
8  |  |  Φ ← Φ ∪ {α ∈ U | ℓ ∈ α};
9  |  |  U ← {α ∈ U | μ(α) = 0};
10 |  |  L ← L ∩ literlas(U);
11 |  else Π ← Π ∪ {¬ℓ} ;
12 end
13 return Σ\Φ;
```

Le schéma algorithmique de l'approche LBX est présenté dans Algorithme 2.6. Initialement, l'algorithme commence par le traitement d'une partition des clauses de Σ en deux partie (Φ, U) au niveau de la procédure approximatePartition. L'ensemble $\Phi \subseteq \Sigma$ contient les clauses satisfaites et l'ensemble $U \subseteq \Sigma$ contient les clauses insatisfaites par une interprétation complète de Σ. Les ensembles $\mathcal{L}$ et Π constituent respectivement l'ensemble des littéraux des clauses de U et les littéraux *backbones* de Φ. Initialement, Π est vide. En entrant dans la boucle *while*, à chaque tour de boucle LBX retire un littéral ℓ de $\mathcal{L}$ et appelle le solveur SAT sur $\Phi \cup \Pi \cup \{\ell\}$. Soulignons que Π n'altère pas le résultat de l'appel au solveur mais permet simplement de simplifier la résolution à l'intérieur du solveur. Si la réponse est SAT, alors les ensembles Φ et U sont mis à jour à partir du modèle calculé par le solveur (lignes $8-9$) et $\mathcal{L}$ est calculé à nouveau. Sinon, $\neg\ell$ est inséré dans l'ensemble des littéraux *backbones* Π. L'algorithme sort de

la boucle lorsque ($\mathcal{L} = \emptyset$), et le MCS final $\Sigma \backslash \Phi$ est retourné. La complexité de LBX est en $\mathcal{O}(|\mathcal{V}|)$ où $\mathcal{V}$ est l'ensemble des variables de l'instance. Enfin, notons que LBX est actuellement la meilleure approche de l'état de l'art pour l'extraction d'un MCS.

Approche RS (Relaxation Search)

RS (*Relaxation Search*) (Bacchus *et al.* 2014) est une approche basée sur la résolution SAT avec des préférences (Rosa *et al.* 2010), dédiée aux problèmes MAX-SAT ou d'extraction des MCSes. Les approches de calcul d'un MCS décrites jusqu'ici utilisent toutes le solveur SAT comme une boîte noire, ce dernier étant appelé de manière incrémentale sur une séquence de formules très similaires (partageant des clauses entre elles) et en fonction de la réponse SAT ou UNSAT des traitements sont effectués. En revanche dans RS, le solveur SAT est modifié de sorte à prendre en compte les préférences sur les littéraux de la formule et de plus il est lancé sur une seule instance. À la fin de la résolution, un modèle est retourné et il constitue la solution finale du problème. Concrètement, RS consiste à générer initialement une formule Σ^S_{eq} à partir de la formule CNF Σ donnée en entrée, telle que pour chaque clause $\alpha_i \in \Sigma$ une variable fraîche s_i lui est associée de la manière suivante $(\neg\alpha_i \leftrightarrow s_i)$. Ainsi, l'ensemble des clauses $(\neg\alpha_i \leftrightarrow s_i)$ forment les clauses de la nouvelle formule Σ^S_{eq}. Un appel au solveur CDCL est effectué sur Σ^S_{eq} et une préférence au niveau de l'heuristique de branchement est donnée aux variables sélecteurs s_i et avec une polarité à 0. L'affectation d'une variable s_i à *faux* permet d'activer la clause α_i attachée à s_i et inversement, l'affectation à *vrai* désactive cette clause. Par ailleurs, l'affectation de s_i à *vrai* n'est possible que si tous les littéraux de la clause associée α_i sont tous à *faux*, autrement dit α_i est falsifiée. De ce fait, le solveur affecte prioritairement les s_i à *faux* afin d'activer un maximum de clauses, ou d'une autre manière, désactiver le moins possible de clauses. Le MCS retourné par l'algorithme correspond alors à l'ensemble des clauses désactivées, c'est-à-dire les clauses associées aux variables s_i affectées à *vrai*.

2.3 Approches pour l'énumération des MSSes/MCSes

L'énumération des MCSes présente un réel défi d'un point de vue calculatoire. La difficulté de cette tâche se justifie par deux raisons :

- calculer un MCS est un problème de complexité $\Delta^P_2 = P^{NP}$ (Papadimitriou 1994a) ;
- dans les pires des cas une intsance CNF insatisfaisable possède un nombre exponentiel de MCSes.

Néanmoins, en pratique ce nombre est souvent relativement bas, permettant ainsi l'énumération complète des MCSes dans les instances.

Les approches d'énumération de MCSes peuvent être réparties en deux catégories : les approches qui calculent tous les MCSes de manière directe et celles qui s'appuient sur la dualité *hitting set* qui existe entre MCSes et MUSes.

2.3.1 Approches directes

L'algorithme 2.7 décrit l'organisation générale des approches directes pour l'énumération des MCSes (Liffiton et Sakallah 2009, Marques-Silva *et al.* 2013, Bacchus *et al.* 2014, Previti *et al.* 2017; 2018). L'algorithme consiste à calculer de manière itérative un MCS de l'instance CNF et puis à « bloquer » ce MCS par une nouvelle clause qu'il génère à partir de la disjonction des littéraux des clauses de ce dernier. La clause bloquante s'assure qu'au moins une des clause du MCS sera activée dans les solutions des

énumérations suivantes. Ainsi un même MCS ne peut être recalculé. Dans l'algorithme 2.7, les clauses bloquantes sont insérées dans Δ (ligne 6). L'algorithme 2.7 termine lorsque $(\Sigma^S \cup \Delta)$ devient UNSAT, dans ce cas tous les MCSes de Σ ont été énumérés.

Algorithme 2.7 : `Enum-MCSes` (Enumerate all MCSes)

input : une formule CNF Σ
output : tous les MCSes de Σ

1 $\Sigma^S = \{\alpha \vee \neg s_\alpha \mid \alpha \in \Sigma\}$; `// avec` s_α `de nouvelles variables`
2 $\Delta \leftarrow \emptyset$;
3 **while** SAT$\Sigma^S \cup \Delta$ == *true* **do**
4 $\quad \Gamma \leftarrow$ `ExtractMCS`$(\Sigma^S \cup \Delta)$;
5 $\quad$ *output*(Γ);
6 $\quad \Delta \leftarrow \Delta \cup (\bigvee_{s_\alpha \in \Gamma} s_\alpha)$; `// clauses bloquantes`
7 **end**

Exemple 2.6. Soit Σ une formule CNF insatisfaisable, telle que $\Sigma = \{\alpha_1 = a \vee b,\ \alpha_2 = \neg a \vee b,\ \alpha_3 = a \vee \neg b, \alpha_4 = \neg a \vee \neg b, \alpha_5 = \neg b, \alpha_6 = b\}$. Déroulons l'algorithme `Enum-MCSes` sur Σ. Initialement, l'algorithme génère la formule CNF augmentée avec les variables sélecteurs $\Sigma^S = \{a \vee b \vee \neg s_1,\ \neg a \vee b \vee \neg s_2,\ a \vee \neg b \vee \neg s_2,\ \neg a \vee \neg b \vee \neg s_2,\ \neg b \vee \neg s_2,\ b \vee \neg s_2\}$. Ensuite, supposons que $\Gamma_1 = \{\alpha_1, \alpha_6\}$, $\Gamma_2 = \{\alpha_2, \alpha_6\}$ et $\Gamma_3 = \{\alpha_3, \alpha_5\}$ soient les premiers MCSes trouvés par la procédure `ExtractMCS`, alors $\Delta = \{s_1 \vee s_6,\ s_2 \vee s_6,\ s_3 \vee s_5\}$. Étant donné $\Sigma^S \cup \Delta$ toujours SAT, l'algorithme effectue une autre itération et le MCS calculé est $\Gamma_4 = \{\alpha_4, \alpha_5\}$. La clause bloquante $(s_4 \vee s_5)$ de Γ_4 est insérée dans Δ. L'algorithme teste la satisfaisabilité de $\Sigma^S \cup \Delta$ et trouve UNSAT (à cette étape la satisfaction de Δ active toutes les clauses de Σ^S) et donc tous les MCSes de Σ ont été énumérés.

Clairement, l'efficacité de cet algorithme dépend fortement des performances de la méthode (`ExtractMCS`) utilisée pour chercher un MCS à chaque itération. À cet égard, (Previti *et al.* 2017) proposent d'utiliser un mécanisme de *caching* dans l'algorithme de recherche linéaire `ELS` (Algorithme 2.2) pour le calcul d'un MCS dans le but d'améliorer l'énumération des MCSes. L'idée consiste à sauvegarder dans une base (*cache*) les CORES retournés par le solveur SAT durant le calcul successif des différents MCSes. L'algorithme 2.2 présente le pseudocode de la méthode de recherche linéaire avec *caching* (`ELS-CACHE`). Dans `ELS-CACHE`, à chaque itération, avant de sélectionner une clause α pour tester si elle est satisfaisable avec le MSS Φ en construction, la base est interrogée afin de savoir si $\Phi \cup \{\alpha\}$ possède un CORE qui a été précédemment trouvé. Si c'est le cas, $\Phi \cup \{\alpha\}$ est UNSAT et donc α est déclarée clause de transition de Φ (par construction Φ est satisfaisable), ce qui économise un appel potentiellement coûteux à l'oracle SAT. Sinon, la formule $\Phi \cup \{\alpha\}$ est testée de manière habituelle en appelant le solveur SAT. Le cache est implémenté dans l'algorithme 2.8 via une formule $\mathcal{D}$ de Horn [5] contenant pour chaque CORE δ trouvé durant le processus d'énumération, une clause $(\bigvee_{\beta_j \in \delta} \neg s_j)$. Pour interroger le *cache* sur la formule $\Phi \cup \{\alpha\}$, un appel en temps polynomial est effectué au solveur SAT sur la formule $\mathcal{D} \cup S$, où S est l'ensemble des sélecteurs correspondant aux clauses de la formule $\Phi \cup \{\alpha\}$.

Très récemment, une autre approche basée sur le concept de *caching* a été introduite dans (Previti *et al.* 2018). Cette fois, l'idée ne consiste plus à garder en *cache* des CORES de la formule insatisfaisable, mais plutôt des « ensembles prémisses » (Kullmann 2011).

Définition 2.8. (Ensemble prémisse (PS))

5. Le problème de satisfaisabilité d'une formule de Horn est en classe *P*.

Soient α une clause et Θ un ensemble de clauses. On dit que Θ est un **ensemble prémisse** (ou PS pour *Premise Set*) de α si et seulement si $\Theta \models \alpha$.

Exemple 2.7. Soit la formule CNF $\Sigma = \{a,\ (\neg a \vee b),\ (\neg c \vee d)\}$. Ici, on a b qui est un littéral impliqué et l'ensemble prémisse qui implique b est l'ensemble des clauses $\{a,\ (\neg a \vee b)\}$. On a aussi a qui est un littéral impliqué et le PS impliquant a est la clause unitaire a elle même.

Algorithme 2.8 : ELS-CACHE (*ELS with caching cores*)

Input : une formule CNF Σ
Output : un MCS de Σ

1 $(\Phi, U) \leftarrow$ DisjointCores(Σ); // Φ est un MSS en construction et U est un ensemble de CORES disjoints $\{U_1, \ldots, U_n\}$.
2 $\Gamma \leftarrow \emptyset$; // Γ est un MCS en construction.
3 **foreach** $U_i \in U$ **do**
4 **while** $U_i \neq \emptyset$ **do**
5 $\alpha \leftarrow$ **choose** $\alpha \in U_i$;
6 $U_i \leftarrow U_i \backslash \{\alpha\}$;
7 $S \leftarrow \{s_j \mid \beta_j \in (\Phi \cup \{\alpha\})\}$;
8 $(res,\ \mu) \leftarrow \mathrm{SAT}(\mathcal{D} \cup S)$;
9 **if** $res = true$ **then**
10 $\Gamma \leftarrow \Gamma \cup \{\alpha\}$;
11 **else**
12 $(res,\ \mu,\ \delta) \leftarrow \mathrm{SAT}(\Phi \cup \Pi \cup \{\alpha\})$;
13 **if** $res = true$ **then**
14 $\Phi \leftarrow \Phi \cup \{\alpha\}$;
15 $\Phi \leftarrow \Phi \cup \{\beta \in U_i \mid \mu(\beta) = 1\}$;
16 $U_i \leftarrow \{\beta \in U_i \mid \mu(\beta) = 0\}$;
17 **else**
18 $\Gamma \leftarrow \Gamma \cup \{\alpha\}$;
19 $\mathcal{D} \leftarrow \mathcal{D}\{(\bigvee_{\beta_j \in \delta} \neg s_j)\}$;
20 **return** Γ;

Le *caching* des ensembles prémisses est intégré dans l'approche LBX (Algorithme 2.6) pour le calcul d'un MCS, donnant lieu à une nouvelle approche LBX-CACHE (Algorithme 2.9). L'objectif est d'éviter des appels non nécessaires au solveur SAT lors du test de satisfaisabilité de la formule $\Phi \cup \{\ell\}$ (Algorithme 2.6 ligne 6). En effet, si Φ est un ensemble prémisse de la clause unitaire $\neg\ell$, alors $\Phi \not\models \ell$, ce qui revient à dire que $\Phi \cup \{\ell\}$ est insatisfaisable. Dans ce cas de figure, un appel au solveur SAT sur la formule $\Phi \cup \{\ell\}$ est utile. Pour tester si Φ est un ensemble prémisse de $\neg\ell$, une requête est envoyée au *cache* pour determiner si il existe un ensemble prémisse $\Theta \subseteq \Phi$ dans le *cache* et qui implique $\neg\ell$ et dans la cas positif, alors Φ est aussi un ensemble prémisse de $\neg\ell$. Cette requête s'effectue au début de chaque itération dans LBX avant d'appeler le solveur SAT. Ainsi, si la réponse de la requête est positive, l'appel au solveur SAT est évité. Les ensembles prémisses à sauvegarder dans le *cache* sont calculés à partir des CORES retournés par le solveur SAT durant les appels successifs de LBX. En effet, lorsque le solveur SAT répond par UNSAT sur la formule $\Phi \cup \{\ell\}$, un CORE Π est fourni par ce dernier. De cette manière, l'ensemble prémisse $\Theta = \Pi \backslash \{\ell\}$ de $\neg\ell$ s'obtient en supprimant simplement ℓ de Π.

Algorithme 2.9 : LBX-CACHE *(LBX with Premise Sets Caching)*

input : une formule CNF Σ
output : un MCS de Σ

```
1  (Φ, U) ← approximatePartition(Σ);
2  L ← literals(U);                    // L est un ensemble de littéraux.
3  Π ← ∅;                              // Π est un ensemble de littéraux backbones.
4  while L ≠ ∅ do
5      ℓ ← RemoveLiteral(L);
6      if Cache.entails(Φ, ¬ℓ) then
7          Π ← Π ∪ {¬ℓ};
8      else
9          (res, μ, δ) ← SAT(Φ ∪ Π ∪ {ℓ});
10         if res = true then
11             Φ ← Φ ∪ {α ∈ U | ℓ ∈ α};
12             U ← {α ∈ U | μ(α) = 0};
13             L ← L ∩ literlas(U);
14         else
15             Π ← Π ∪ {¬ℓ};
16             Θ ← δ\{ℓ};              // Θ est un ensemble prémisse de ¬ℓ.
17             Cache.add(Θ, ¬ℓ);
18         end
19     end
20 end
21 return Σ\Φ;
```

2.3.2 Approches basées sur la dualité *hitting set*

L'énumération des MCSes peut se faire d'une autre manière que celle décrite dans la section 2.3.1 (Algorithme 2.7) en se basant sur la connexion qui existe entre les deux concepts MCSes et MUSes. En fait, il est possible de déduire l'ensemble de MCSes d'une formule insatisfaisable à partir de l'ensemble des ses MUSes et *vice versa*. Cette dualité entre l'ensemble des MCSes et l'ensemble des MUSes est basée sur le concept de « *hitting set* ».

Définition 2.9. (*Hitting Set* (HS))
Soit une collection d'ensembles Λ construits sur un domaine fini $\mathcal{D}$, un ensemble intersectant ou ***hitting set*** (HS) de Λ est un ensemble d'éléments de $\mathcal{D}$ qui possède au moins un élément commun à chaque ensemble de Λ. Formellement :

$\mathcal{H} \subseteq \mathcal{D}$ est un ***hitting set***, noté HS, de Λ si et seulement si $\forall \lambda_i \in \Lambda$, $(\mathcal{H} \cap \lambda_i) \neq \emptyset$

De plus, $\mathcal{H}$ est dit **minimal**, noté MHS, si aucun sous-ensemble de $\mathcal{H}$ n'est un *hitting set* de Λ.

La dualité qui existe entre les deux concepts MCSes et MUSes a été mise en évidence pour la première fois en intelligence artificielle par Reiter dans (Reiter 1987) et de Kleer et Williams dans (de Kleer et Williams 1987), et elle est basée sur la notion de MHS.

Définition 2.10. (Dualité MCS-MUS)
Soient une formule CNF insatisfaisable Σ et les sous-ensembles $\Psi \subseteq \Sigma$ et $\Upsilon \subseteq \Sigma$, alors :

- Ψ est un MCS de Σ si et seulement si Ψ est un MHS de la collection des MUSes de Σ ;

- Υ est un MUS de Σ si et seulement si Υ est un MHS de la collection des MCSes de Σ.

Rappelons que la présence d'un MUS dans une formule CNF Σ rend cette dernière incohérente. Par définition, un MUS peut être rendu satisfaisable en lui supprimant une de ses clauses. Par conséquent, une manière de rétablir la cohérence dans Σ consisterait à rectifier l'ensemble des ses MUSes en supprimant au moins une clause dans chacun d'eux. Un MCS est un ensemble irréductible de clauses à supprimer pour rendre Σ satisfaisable. De ce fait, chaque MCS contient au moins une clause de chaque MUS de Σ. Ainsi, de manière intuitive on peut voir que l'ensemble des MCSes est équivalent à l'ensemble des minimaux *hitting sets* des MUSes. De façon similaire, on peut montrer que l'ensemble des MUSes correspond à l'ensemble des MHSes des MCSes.

Cette relation est illustrée dans l'exemple 2.8.

Exemple 2.8. Considérons la formule CNF Σ de l'exemple 2.6. La Table 2.1 correspond au problème de recherche des *hitting sets* de l'ensemble des MCSes pour générer les MUSes de Σ, tandis que la Table 2.2 correspond au problème dual de recherche des *hitting sets* de l'ensemble des MUSes pour générer les MCSes de Σ. Dans le première table, chaque colonne correspond à une clause de Σ et chaque ligne représente un MCS. On dit qu'une clause *couvre* un MCS (marqué avec une croix '$\times$' dans la ligne) si elle est incluse dans le MCS. Chaque MUS est alors un sous-ensemble irréductible des colonnes qui couvre toutes les lignes. La deuxième table représente, de la même manière, les MUSes et chaque MCS est un sous-ensemble irréductible des colonnes qui couvre toutes les lignes des ces MUSes. Au dessous de chaque table, nous montrons comment les MUSes peuvent être calculés à partir de la table des MCSes et des MCSes de la table des MUSes, de manière simple : chaque ligne devient une disjonction des colonnes qui couvrent cette ligne, et les disjonctions sont multipliées et simplifiées en supprimant les termes subsumés pour obtenir les MHSes.

MCSes	α_1	α_2	α_3	α_4	α_5	α_6
$\{\alpha_1, \alpha_6\}$	$\times$					$\times$
$\{\alpha_2, \alpha_6\}$		$\times$				$\times$
$\{\alpha_3, \alpha_5\}$			$\times$		$\times$	
$\{\alpha_4, \alpha_5\}$				$\times$	$\times$	

TABLE 2.1 – Ensemble des MCSes de la formule CNF Σ de l'exemple 2.6.

$$\begin{aligned} \text{MUS}es(\Sigma) &= (\alpha_1 \vee \alpha_6)\,(\alpha_2 \vee \alpha_6)\,(\alpha_3 \vee \alpha_5)\,(\alpha_4 \vee \alpha_5) \\ &= \alpha_1\alpha_2\alpha_3\alpha_4 \vee \alpha_1\alpha_2\alpha_5 \vee \alpha_3\alpha_4\alpha_6 \vee \alpha_5\alpha_6 \\ &= \{\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}, \{\alpha_1, \alpha_2, \alpha_5\}, \{\alpha_3, \alpha_4, \alpha_6\}, \{\alpha_5, \alpha_6\}\}. \end{aligned}$$

MUSes	α_1	α_2	α_3	α_4	α_5	α_6
$\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$	$\times$	$\times$	$\times$	$\times$		
$\{\alpha_1, \alpha_2, \alpha_5\}$	$\times$	$\times$			$\times$	
$\{\alpha_3, \alpha_4, \alpha_6\}$			$\times$	$\times$		$\times$
$\{\alpha_5, \alpha_6\}$					$\times$	$\times$

TABLE 2.2 – Ensemble des MUSes de la formule CNF Σ de l'exemple 2.6.

$$\begin{aligned} \text{MCS}es(\Sigma) &= (\alpha_1 \vee \alpha_2 \vee \alpha_3 \vee \alpha_4)\,(\alpha_1 \vee \alpha_2 \vee \alpha_5)\,(\alpha_3 \vee \alpha_4 \vee \alpha_6)\,(\alpha_5 \vee \alpha_6) \\ &= \alpha_1\alpha_6 \vee \alpha_2\alpha_6 \vee \alpha_3\alpha_5 \vee \alpha_4\alpha_5 \\ &= \{\{\alpha_1, \alpha_6\}, \{\alpha_2, \alpha_6\}, \{\alpha_3, \alpha_5\}, \{\alpha_4, \alpha_5\}\}. \end{aligned}$$

La dualité entre MCSes et MUSes permet d'établir un algorithme d'énumération de MCSes. Cet algorithme prend en entrée la collection de MUSes de la formule insatisfaisable et calcule de manière itérative les MCSes de la formule et par le biais d'une procédure d'extraction d'un MHS appelé à chaque itération.

Dans (Reiter 1987), une approche similaire a été proposée pour le calcul de diagnostics d'erreurs dans un système de circuits. Le problème est encodé en logique du premier ordre fini et les modèles des MCSes à identifier correspondent aux diagnostics dans le système.

L'approche pour l'énumération des MCSes basée sur la notion de dualité *hitting set* semble être une alternative par rapport à l'approche directe. Toutefois, en pratique les techniques actuelles de l'état de l'art s'appuient sur le schéma algorithmique décrit dans `Enum-MCSes` (Algorithme 2.7). En revanche, il existe un nombre important de méthodes pour l'énumération de MUSes basées sur la dualité *hitting set* (voir par exemple (Bailey et Stuckey 2005, Grégoire *et al.* 2007b, Liffiton et Sakallah 2008, Nöhrer *et al.* 2012, Previti et Marques-Silva 2013, Liffiton *et al.* 2016, Bacchus et Katsirelos 2015; 2016, Narodytska *et al.* 2018), etc).

2.4 Ensemble maximal satisfaisable avec une série de contextes

Dans (Besnard *et al.* 2015), une variante du concept de MSS est introduite. Cette variante concerne le sous-ensemble maximal satisfaisable d'un ensemble de clauses Σ sous une série de contextes hypothétiques $\mathcal{AC}$, noté AC-MSS (*Maximal Satisfiable Subset of* Σ *under a set of Assumptive Contexts* $\mathcal{AC}$).

Le calcul d'un sous-ensemble maximal de clauses qui soit satisfaisable avec des contextes multiples ou des sources d'informations supplémentaires (potentiellement contradictoires), représentés sous forme clausale, est un problème clé dans de nombreux domaines de l'Intelligence Artificielle, en particulier, lorsque ces contextes ou sources peuvent être mutuellement contradictoires. Comme illustré dans (Besnard *et al.* 2015), ce problème est central en planification, dans la prise de décision, le diagnostic d'erreur, l'argumentation et le raisonnement non monotone, ainsi que dans bien d'autres champs de l'Intelligence Artificielle.

Considérons par exemple un agent qui doit prendre des décisions extrêmement rapidement à partir de ses propres informations. Il souhaite être prudent et ne prendre que des décisions qui soient compatibles avec un éventail d'hypothèses prospectives, possiblement contradictoires, qu'il pourrait envisager au sujet de ce qu'il ne sait pas. En raison des contraintes de temps, il cherche à extraire rapidement un sous-ensemble maximal d'informations qui ne contredit aucune de ces hypothèses et considère ce sous-ensemble comme formant une base acceptable pour sa prise de décision. De cette manière, chaque hypothèse restera compatible avec ses propres décisions et la prise en compte supplémentaire de n'importe quelle autre de ses propres prémisses entrainera une contradiction avec au moins une de ces hypothèses. Cette extraction pourrait être aussi à la base d'une méthode d'énumération des sous-ensembles maximaux cohérents avec toutes les hypothèses lorsque ces sous-ensembles demeurent en petit nombre et qu'un temps de calcul suffisant est disponible. Elle peut également être adaptée de façon à respecter un ordre de préférence entre informations.

De manière générale, l'extraction d'un sous-ensemble maximal d'informations cohérent avec des sources d'informations supplémentaires, sources pouvant être mutuellement contradictoires, peut jouer un rôle dans les systèmes multi-agents et la négociation, car cela peut consister à trouver un sous-ensemble maximal qui n'entre en conflit avec aucun des objectifs fixés par les agents ou les négociateurs.

2.4.1 Définitions

Nous reprenons de (Besnard *et al.* 2015) les définitions de sous-ensembles maximaux satisfaisables et minimaux rectificatifs *sous un ensemble de contextes hypothétiques*.

Soient Σ est un ensemble fini de clauses, Φ et Ψ sont des sous-ensembles de Σ et $\mathcal{AC}$ est un ensemble fini de contextes, notés Γ_i, tels que chaque Γ_i est un ensemble satisfaisable de clauses.

Définition 2.11. **(AC-MSS)**
$\Phi \subseteq \Sigma$ est un sous-ensemble maximal satisfaisable de Σ sous un ensemble de contextes hypothétiques $\mathcal{AC}$, noté AC-MSS$(\Sigma, \mathcal{AC})$, si et seulement si :

- $\forall \Gamma_i \in \mathcal{AC}$, $\Phi \cup \Gamma_i$ est satisfaisable ;
- $\forall \alpha \in \Sigma \setminus \Phi$, $\Phi \cup \{\alpha\} \cup \Gamma_i$ est insatisfaisable pour certains $\Gamma_i \in \mathcal{AC}$.

Exemple 2.9. Soient $\Sigma = \{a \vee b, a \vee c, d \vee b, e \vee c, \neg b, \neg c\}$ et $\mathcal{AC} = \{\{\neg a\}, \{\neg d\}, \{\neg e\}\}$. Soulignons que cet exemple est un cas simple dans le sens où (1) Σ est satisfaisable, (2) tous les contextes sont de simples clauses unitaires et (3) les contextes ne sont pas mutuellement contradictoires. Dans le cas général ces trois propriétés n'ont pas besoin d'être satisfaites. Dans cet exemple, la figure 2.2 illustre comment les différents contextes de $\mathcal{AC}$ entrent en conflit avec les clauses de Σ. Les ensembles de clauses regroupés dans des cercles représentent les MUSes formés par $\Sigma \cup \mathcal{AC}$, avec à chaque fois un contexte de $\mathcal{AC}$. Dans cet exemple, $\{a \vee b,\ a \vee c,\ d \vee b,\ e \vee c\}$ constitue un AC-MSS$(\Sigma, \mathcal{AC})$, parmi d'autres.

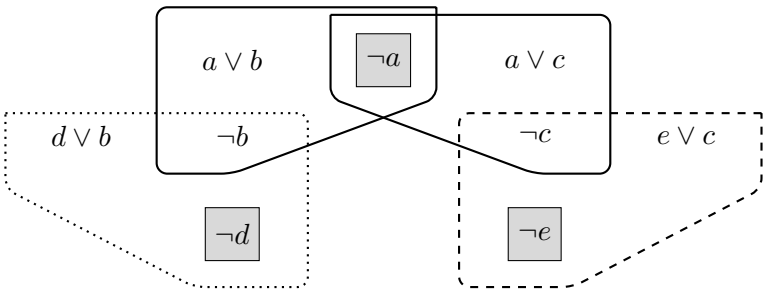

FIGURE 2.2 – Ensemble des MUSes formés par $\Sigma \cup \mathcal{AC}$ de l'exemple 2.9.

Définition 2.12. **(AC-MCS)**
$\Psi \subseteq \Sigma$ est un sous-ensemble minimal rectificatif de Σ sous un ensemble de contextes hypothétiques $\mathcal{AC}$, noté AC-MCS$(\Sigma, \mathcal{AC})$, si et seulement si $\Psi = \Sigma \setminus \Phi$, où Φ est un AC-MSS$(\Sigma, \mathcal{AC})$.

Exemple 2.10. Dans l'exemple 2.9, $\{a \vee b,\ a \vee c,\ d \vee b,\ e \vee c\}$, $\{a \vee b,\ d \vee b,\ \neg c\}$, $\{a \vee c,\ e \vee c,\ \neg b\}$ et $\{\neg c,\ \neg b\}$ sont des AC-MCS$(\Sigma, \mathcal{AC})$.

Les approximations de AC-MSS$(\Sigma, \mathcal{AC})$ et AC-MCS$(\Sigma, \mathcal{AC})$ sont définies comme suit.

Définition 2.13. **(AC-SS)**
On dit que Φ est un *sous-ensemble satisfaisable* de Σ sous un ensemble de contextes hypothétiques $\mathcal{AC}$, noté AC-SS$(\Sigma, \mathcal{AC})$, si est seulement si il existe au moins un ensemble de clauses Φ' tel que $\Phi' \subseteq \Phi \subseteq \Sigma$ et Φ' est un AC-MSS$(\Sigma, \mathcal{AC})$

Définition 2.14. **(AC-CS)**
On dit que Ψ est un *sous-ensemble rectificatif* de Σ sous un ensemble de contextes hypothétiques $\mathcal{AC}$, noté AC-CS$(\Sigma, \mathcal{AC})$, si et seulement si il existe au moins un ensemble de clauses Ψ' tel que $\Psi' \subseteq \Psi \subseteq \Sigma$ et Ψ' est un AC-MCS$(\Sigma, \mathcal{AC})$.

Insistons sur le fait que ni Σ, ni la conjonction de tous les Γ_i de $\mathcal{AC}$, ne doivent être forcément satisfaisables dans les définitions. Les Γ_i sont considérés un à un : chaque AC-MSS$(\Sigma, \mathcal{AC})$ doit être satisfaisable avec chaque contexte Γ_i, sélectionné individuellement. Dans (Besnard *et al.* 2015), des exemples illustrent pourquoi les Γ_i ne peuvent pas être remplacés par un contexte globalisé qui soit formé par leur conjonction ou leur disjonction, même dans les cas où aucun Γ_i n'en contredit aucun autre.

2.4.2 Calcul d'un AC-MSS/AC-MCS

La méthode, dite "par transformation" (`Transformational-AC-MSS`), pour l'extraction d'un AC-MSS$(\Sigma, \mathcal{AC})$ que proposent (Besnard *et al.* 2015), est la plus performante qui existe dans la littérature. Elle fait appel à une procédure de pré-traitement (`Greedy-AC-SS`), qui permet d'extraire un sous-ensemble d'un AC-MSS$(\Sigma, \mathcal{AC})$, c'est-à-dire, un AC-SS$(\Sigma, \mathcal{AC})$ de départ qu'il faudra maximiser par la suite. Concrètement, `Greedy-AC-SS` consiste à prendre un ensemble de clauses Σ' initialisé à Σ, lequel va être réduit de manière itérative jusqu'à ce que Σ' devienne satisfaisable avec tous les Γ_i. En effet, à chaque itération, un Γ_i est considéré, les clauses satisfaisable avec Γ_i sont gardées et les autres sont ainsi supprimées.

Exemple 2.11. Soit $\Sigma = \{\neg a \vee b, \neg b, b \vee d\}$ et $\mathcal{AC} = \{\Gamma_1 = \{a\}, \Gamma_2 = \{\neg d\}\}$. `Greedy-AC-SS` se déroule comme suit. Supposons que Γ_1 soit considéré en premier et $\neg a \vee b$ soit sélectionné et retiré de Σ à cette étape. Ensuite, admettons que $\neg b$ soit supprimé afin de rendre Σ' satisfaisable avec Γ_2. L'ensemble final $\Sigma' = \{b \vee d\}$ n'est pas un AC-MSS$(\Sigma, \mathcal{AC})$. En effet, seule la suppression de $\neg b$ de Σ permettra d'obtenir un plus grand ensemble $\{\neg a \vee b, b \vee d\}$, lequel est un AC-MSS$(\Sigma, \mathcal{AC})$.

Dans (Besnard *et al.* 2015), une approche de *force brute* d'extraction d'un AC-MSS$(\Sigma, \mathcal{AC})$ est analysée. Cette approche consiste à calculer pour chaque Γ_i tous les sous-ensembles maximaux de Σ cohérents avec Γ_i, avant d'extraire la solution finale qui minimise les clauses écartées. Le problème majeur de cette méthode est le nombre potentiellement exponentiel de MSS qu'il peut y avoir à calculer, rendant pareille approche intraitable le plus souvent.

Pour pallier cette difficulté, du moins dans une certaine mesure, les auteurs ont proposé dans (Besnard *et al.* 2015) un schéma d'encodage qui permet de réécrire le problème en un seul calcul de MSS avec quelques appels de plus à un solveur SAT. Précisément, quand m est le nombre de Γ_i dans $\mathcal{AC}$, n' le nombre maximal de clauses dans un Γ_i, n le nombre de clauses dans Σ, la méthode nécessite dans le pire cas $\mathcal{O}(m)$ appels à un solveur SAT sur une instance de taille $\mathcal{O}(n + n')$, plus un nombre logarithmique d'appels à un solveur SAT sur une instance de taille initiale $\mathcal{O}(m(n + n'))$ qui est divisée par deux à chaque appel. Sans surprise, `Transformational-AC-MSS` s'avère plus efficace que la méthode *force-brute* dans beaucoup d'instances.

2.5 Conclusion

Le problème de recherche des sous-ensembles maximaux satisfaisables (MSSes) dans le cadre de la logique propositionnelle a fait l'objet de nombreuses études ces dernières années. Des avancées significatives dans la résolution pratique de ce problèmes ont ainsi pu être observées, à la fois dans le but de calculer un MSS que dans l'enumerations des MSSes. Dans la deuxième partie de ce document, nos différentes contributions sur le calcul de tels ensembles sont dépeintes.

Chapitre

3

Consensus

Sommaire

La définition d'un **consensus** telle qu'elle est donnée dans Wikipédia est la suivante : *"Un consensus est un accord des volontés sans aucune opposition formelle"*. Illustrons ceci avec exemple concret où une forme de consensus est recherché. Supposons que trois groupes politiques négocient un éventuel programme de coalition en vue de former un gouvernement, alors que leurs programmes politiques sont mutuellement contradictoires. Pour ce faire, un consensus est recherché pour dégager un certain nombre de points (idées/objectifs/réformes) parmi toutes les propositions offertes par les trois programme, de sorte qu'aucun d'eux ne soit désavoué/contesté. Bien qu'un groupe politique puisse trouver dans le consensus des éléments qui n'appartiennent pas à son propre programme, il pourrait tout à fait approuver ce consensus puisque ces éléments ne contredisent pas ses propres objectifs.

En intelligence artificielle, la définition d'une forme de consensus constitue un réel paradigme. Par exemple, dans un groupe d'agents interactifs (Ephrati et Rosenschein 1996, Ren *et al.* 2005), un consensus peut prendre la forme d'un arrangement unanime auquel aboutit le groupe après négociation. Dans la dynamique des croyances (Jøsang 2002, Gauwin *et al.* 2007), le consensus peut être par exemple le résultat final d'un processus de conciliation d'agents intelligents à propos de leurs croyances. Plus généralement, la notion de consensus est une manière de réconcilier des opinions, des connaissances, des croyances, des désirs, des plans, des objectifs conflictuelles ou de manière globale, des sources d'informations contradictoires.

Cependant, les concepts de consensus sont rarement définis de manière précise, notamment dans le contexts où les agents sont dotés d'une capacité de déduction. Dans (Grégoire *et al.* 2016c), une notion de consensus a été définie dans le contexte de la logique propositionnelle. Les sources d'informations sont représentées par des ensembles de formules logiques et le consensus est défini comme étant un ensemble de formules appartenant aux sources et qui soit cohérent avec chaque source prise de manière indépendante. De plus, un tel consensus, pourrait ne pas contenir uniquement les informations partagées par toutes les sources ; il peut aussi contenir des informations supplémentaires qui sont dans un certain sens possiblement acceptables du point de vue de chaque source car elles ne sont contredites par aucune source. Par conséquent, chaque source pourrait approuver les informations contenue dans le consensus car cette information appartient aussi à la source ou n'est pas en conflit avec celle-ci. Il est noté aussi dans (Grégoire *et al.* 2016c) que certains consensus peuvent être préférés à d'autres, en conséquence plusieurs familles de critères de préférence sont introduites et peuvent être utilisées pour sélectionner des consensus.

La suite de ce chapitre est organisée comme suit. D'abord, une brève présentation sur la dynamique des croyances est donnée et ensuite sont présentées les différentes notions de consensus établies dans

les bases de connaissances représentant des croyances. Ensuite, nous présentons le concept de consensus dans le cadre de la logique propositionnelle. Nous rappelons la définition formelle d'un consensus (dans le cadre propositionnel), exposerons des différents critères de préférences sur les consensus et présenterons l'approche générique de calcul de consensus proposée par Grégoire *et al.* (2016c). Enfin, nous clôturerons ce chapitre avec une conclusion.

3.1 Dynamique des croyances

L'étude de la dynamique des croyances est nécessaire pour pouvoir utiliser et gérer des croyances qui par nature sont incertaines et/ou incomplètes. Les cadres traitant cette dynamique des croyances sont essentiellement la *théorie de la révision* et la *théorie de la fusion* des croyances. La révision de croyances pose le problème suivant : étant donnée une base de connaissances représentant les croyances d'un agent à propos du monde et une nouvelle information apprise sur le monde, potentiellement en conflit avec certaines informations dans la base, quelles modifications doivent être opérées dans celle-ci pour incorporer la nouvelle information ? Un problème similaire est posé lorsqu'il s'agit de plusieurs agents ayant des croyances mutuellement incompatibles et on désire définir une base de croyances reflétant les croyances du groupe d'agents. On parle alors de la fusion des croyances.

Le consensus est un concept qui prend différentes formes dans la dynamiques des croyances. Par exemple, dans (Jøsang 2002), le consensus représente une fonction (opérateur) permettant de combiner des croyances incertaines potentiellement contradictoires.

Dans (Gauwin *et al.* 2007), une notion de conciliation et une notion de consensus sont introduites dans la fusion itérative des croyances. Le problème étudié concerne la façon dont le résultat de la fusion des croyances peut être exploité par le groupes d'agents. Le scénario dressé dans cette étude, concerne des agents très intelligents pour qui il semble improbable d'accepter d'abandonner leurs propres bases de croyances à chacun et d'adopter la base de fusion comme nouvelle base de croyances et ainsi partager tous la même. Il semble plus approprié pour eux d'incorporer le résultat du processus de fusion dans leur base de croyances actuelle. Une telle incorporation de nouvelles croyances nécessite une révision des croyances. (Gauwin *et al.* 2007) ont introduit alors un processus de conciliation des croyances des agents qui peut être vu comme une forme simple de négociation. La méthode consiste à fusionner puis réviser de manière itérative les croyances des agents jusqu'à atteindre un certain niveau d'arrangement entre les agents ; à ce stade chaque agent a fait de son mieux, par rapport au groupe, compte tenu de sa fonction de révision. Au final, deux possibilités se présentent : soit un consensus a été obtenu, soit aucun consensus ne peut être obtenu de cette façon. L'existence de consensus est déterminé lorsque la conjonction de toutes les bases de croyances forme un ensemble cohérent.

3.2 Consensus dans le cadre propositionnel

Les consensus peuvent être définis de différentes manière et respectent des propriétés variées. Dans le cadre de la logique propositionnelle, une forme de consensus a été introduite par (Grégoire *et al.* 2016c) et s'applique sur des sources d'informations (positions/plans/désires/objectifs/connaissances/croyances) encodées en formules propositionnelles. De plus, sont introduites plusieurs familles de critères de préférence qui peuvent être utilisées pour sélectionner des consensus.

3.2.1 Définitions

Techniquement, un *opérateur consensus* est défini (en langage propositionnel) de la manière suivante. Étant donné un ensemble de n sources d'informations $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$, appelé *profile*, où chaque Φ_i est un ensemble de formules satisfaisable représentant la source d'informations de l'agent i ($i \in (1, \ldots, n)$). Un consensus de $\mathcal{S}$ est un sous-ensemble de $\bigcup_{i=1}^{n} \Phi_i$ qui soit cohérent avec chaque Φ_i. Soulignons qu'il est exclu ici de considérer le cas où une source Φ_i est incohérente car ceci conduirait à l'absence de consensus (aucun ensemble de formules ne peut être satisfaisable avec un ensemble de formules insatisfaisable). Toutefois, une source Φ_i peut être ou ne pas être logiquement contradictoire avec une autre source Φ_j ($\Phi_i \cup \Phi_j$ peut être insatisfaisable).

Soient $\mathcal{L}$ le language de la logique propositionnelle et $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$ un profile représentant n sources Φ_i, où chaque $\Phi_i \subset \mathcal{L}$ est un ensemble satisfaisable de formules propositionnelles.

Définition 3.1. (Consensus)
Un ensemble $\Gamma \subset \mathcal{L}$ est un consensus de $\mathcal{S}$ si et seulement si :

- $\Gamma \subseteq \bigcup_{i=1}^{n} \Phi_i$;
- $\forall \Phi_i \in \mathcal{S}$, $\Gamma \cup \Phi_i$ est satisfaisable.

Clairement, un consensus Γ est un ensemble de formules satisfaisable.

Remarque 3.1. Puisque chaque source Φ_i est satisfaisable, alors il existe forcément au moins un consensus pour $\mathcal{S}$, qui peut être l'ensemble vide.

Un premier critère de préférence sur les consensus défini par (Grégoire *et al.* 2016c) concerne la maximalité soit par rapport à l'inclusion ensembliste ($\subseteq$) ou bien la cardinalité ($\#$) .

Définition 3.2. (Consensus $max_{\subseteq}$)
Un consensus Γ de $\mathcal{S}$ est maximal par rapport l'inclusion ensembliste, noté $max_{\subseteq}$, si et seulement si $\forall \Theta$ tel que $\Gamma \subset \Theta \subseteq \bigcup_{i=1}^{n} \Phi_i$, $\exists \Phi_i \in \mathcal{S}$ tel que $\Theta \cup \Phi_i$ est insatisfaisable.

Définition 3.3. (Consensus $max_{\#}$)
Un consensus Γ de $\mathcal{S}$ est maximal par rapport à la cardinalité, noté $max_{\#}$ si et seulement si $\forall \Theta$ tel que $\Gamma \subset \Theta \subseteq \bigcup_{i=1}^{n} \Phi_i$ et $|\Theta| > |\Gamma|$, $\exists \Phi_i \in \mathcal{S}$ tel que $\Theta \cup \Phi_i$ est insatisfaisable.

Notation 3.1. On utilise la notation *consensus maximal* lorsque il n'est pas besoin de différencier entre $max_{\subseteq}$ et $max_{\#}$.

Clairement, tout consensus $max_{\#}$ est un consensus $max_{\subseteq}$, par contre l'inverse n'est pas toujours vrai. Lorsque $\bigcup_{i=1}^{n} \Phi_i$ est satisfaisable, alors $\bigcup_{i=1}^{n} \Phi_i$ est l'unique consensus maximal de $\mathcal{S}$. Il est intéressant de voir que le concept de consensus tel que défini par (Grégoire *et al.* 2016c) est très proche du concept de MSS, mais ils sont différents. En effet, un MSS de $\bigcup_{i=1}^{n} \Phi_i$ n'est pas nécessairement satisfaisable avec chaque Φ_i alors que le consensus l'est, par conséquent un MSS de $\bigcup_{i=1}^{n} \Phi_i$ n'est pas toujours un consensus, et il de même pour un consensus. En définitive, l'ensemble des MSSes et l'ensemble des consensus sont possiblement disjoints.

Exemple 3.1. Reprenons l'exemple précédent, donné en introduction de ce chapitre, sur la négociation de trois groupes politiques pour former une coalition dans le gouvernement. Considérons $\{ai, rss, add\}$ un ensemble de variables booléennes désignants respectivement : *augmenter les impôts*, *réduire la sécurité sociale* et *augmenter les dépenses de la défense*. Soit $\mathcal{S}$ représentant les programme politiques des trois

groupes : $\mathcal{S} = [\Phi_1, \Phi_2, \Phi_3]$ avec $\Phi_1 = \{ai, \neg rss, \neg ai \rightarrow \neg add\ \}$, $\Phi_2 = \{rss, ai \rightarrow add\}$ et $\Phi_3 = \{\neg add\}$.
Il y a trois consensus $max_{\#}$ pour $\mathcal{S}$: $\Gamma_1 = \{\neg ai \rightarrow \neg add, ai \rightarrow add\}$, $\Gamma_2 = \{\neg add, \neg ai \rightarrow \neg add\}$ et $\Gamma_3 = \{ai, \neg ai \rightarrow \neg add\}$. Par exemple Γ_3 dit "Augmentons les impôts et si nous n'augmentons pas les impôts, nous n'augmentons pas les dépenses de défense".

(Grégoire et Lagniez 2016a) étend le concept de consensus en introduisant des formes de contraintes d'intégrité qui doivent être satisfaites par le résultat du consensus. Formellement, un consensus sous contraintes d'intégrités est défini de la manière suivante.

Définition 3.4. (Consensus sous contraintes d'intégrités)
Un ensemble $\Gamma \subset \mathcal{L}$ est un consensus de $\mathcal{S}$ sous contraintes d'intégrités Δ si et seulement si :

- Γ est un consensus de $\mathcal{S}$;
- $\forall \Phi_i \in \mathcal{S}, \Gamma \cup \Phi_i \cup \Delta$ est satisfaisable.

Exemple 3.2. Dans l'exemple 3.1, $\Gamma = \{ai, \neg ai \rightarrow \neg add\}$ est un consensus sous la contrainte d'intégrité $\Delta = \{ai\}$. Par exemple, il n'existe pas de consensus pour $\mathcal{S}$ sous la contrainte d'intégrité $\Delta = \{\neg rss\}$ car rss est en conflit avec Φ_2.

Comme nous l'avons mentionné précédemment, plusieurs familles de critères de préférence sur les consensus ont été introduites dans (Grégoire *et al.* 2016c). Jusqu'à maintenant, nous avons parlé uniquement du critère de maximalité (par rapport à $\subseteq$ ou $\#$). Le second critère proposé concerne le nombre de concepts que le consensus contient. Les concepts sont exprimés dans $\mathcal{S}$ par des variables booléennes. Ainsi, une préférence est accordée au consensus ayant le plus grand nombre de variables booléennes et donc faisant référence à un maximum de concepts. Un troisième critère est défini sur le nombre de sources qui sont totalement satisfaites. Le consensus Γ préféré alors, est celui qui satisfait totalement un maximum de sources Φ_i, et Γ satisfait totalement une source Φ_i si et seulement si $\Phi_i \subseteq \Gamma$. Pour les deux dernières critères de préférences, on utilise un pré-ordre sur les formules de $\bigcup_{i=1}^{n} \Phi_i$ et un autre un pré-ordre sur les sources Φ_i. Dans le chapitre 7, nous présenterons un autre critère de préférence qui porte sur la notion de *consensus admissible*.

Dans le suite de cette section, nous nous focaliserons sur le coté calculatoire d'un consensus.

Il est important de noter que le nombre de consensus possibles pour $[\Phi_1, \ldots, \Phi_n]$ est exponentiel dans le pire cas. Cependant, dans de nombreux problèmes et applications de négociation où un consensus doit être trouvé (rapidement), l'extraction d'un seul consensus maximal est suffisant pour résoudre le problème. Cela peut être également un point de départ utile pour les négociations futures.

3.2.2 Approche générique pour le calcul de consensus

Une approche générique pour le calcul de consensus en fonction des différents critères de préférences a été proposée dans (Grégoire *et al.* 2016c). Nous nous intéressons ici au calcul d'un consensus $max_{\#}$. Le schéma algorithme de la méthode pour le calcul d'un consensus $max_{\#}$ est décrite dans l'algorithme 3.1.

L'extraction d'un consensus $max_{\#}$ revient à calculer une variante du problème MAX-SAT de ($\bigcup_{i=1}^{n} \Phi_i$). En effet, pour rechercher un consensus $max_{\#}$, le calcul d'un MAX-SAT doit également tenir compte de la contrainte supplémentaire exigeant que le résultat soit satisfaisant avec chaque Φ_i considéré individuellement. Notons que puisque les Φ_i peuvent être mutuellement conflictuels, dans le cas général, il n'est pas possible de remplacer simplement cette contrainte multiple par une contrainte unique indiquant

que le résultat doit être satisfaisable avec $\bigcup_{i=1}^{n} \Phi_i$. La méthode introduite dans (Grégoire *et al.* 2016c) repose sur la transformation du problème initial en un problème d'optimisation (MAX-SAT). Intuitivement, la satisfaisabilité de Γ (consensus à construire) avec un Φ_i donné, est interprétée comme un sous-problème. L'ensemble des sous-problèmes sont ensuite liés en utilisant des nouvelles variables fraîches, appelées variables de liaison. L'utilisation d'un solveur (Partial) Max-SAT permet alors d'extraire un sous-ensemble de clauses qui constitue une solution au problème initial.

Algorithme 3.1 : Calcul d'un consensus $max_{\#}$ pour $\mathcal{S}$.

Input : $\mathcal{S} = \{\Phi_1, \ldots, \Phi_n\}$: un ensemble de n ensembles satisfaisables de clauses;
Admettons que les clauses de Φ_i sont désignées par $\alpha_i^1, \alpha_i^2, \ldots$;

Output : un consensus $max_{\#}$ Γ de $\mathcal{S}$;

1 $\Sigma_1 \leftarrow \emptyset$; $\Sigma_2 \leftarrow \emptyset$;
2 $\Omega \leftarrow \bigcup_{\Phi_i \in \mathcal{S}} \{\neg\epsilon_i^j \vee \alpha_i^j$ s.t. $\alpha_i^j \in \Phi_i$ et où ϵ_i^j sont des variables fraîches$\}$;
3 $\Sigma_2 \leftarrow \{\epsilon_i^j\}_{i,j}$;
4 **foreach** $\Phi_i \in \mathcal{S}$ **do**
5 $\quad \Phi_i \leftarrow \Omega \cup \Phi_i$;
6 $\quad$ Renommer toutes les variables dans Φ_i (sauf les ϵ_i^j) avec de nouvelles variables fraîches;
7 $\quad \Sigma_1 \leftarrow \Sigma_1 \cup \Phi_i$;
8 $\Psi \leftarrow \text{Partial-MaxSAT}(\Sigma_1, \Sigma_2)$;
9 $\Gamma \leftarrow \{\alpha_i^j \in \bigcup_{i=1}^{n} \Phi_i$ s.t. $\epsilon_i^j \in \Psi\}$;
10 **return** Γ;

L'algorithme 3.1 prend en entrée un ensemble de sources $\mathcal{S}$, où chaque formule Φ_i est mise sous forme CNF, et renvoie un consensus $max_{\#}$ Γ. Le problème d'avoir Γ étant un sous-ensemble de $\bigcup_{i=1}^{n} \Phi_i$ qui soit satisfaisable avec chaque Φ_i est d'abord traité comme n sous-problèmes indépendants. Ces sous-problèmes seront ensuite connectés pour former un seul problème d'optimisation à résoudre en un seul appel à `Partial-MaxSAT`. Chaque clause δ_i^j de chaque Φ_i est augmentée par un littéral (sélecteur) $\neg\epsilon_i^j$ telle que ϵ_i^j est une nouvelle variable (ligne 2); les clauses $\delta_i^j \vee \neg\epsilon_i^j$ sont insérées dans Ω. Chaque sous-problème est crée par l'union de Ω et Φ_i et en renommant toutes les variables dans $\Omega \cup \Phi_i$ sauf les variables sélecteurs ϵ_i^j (ligne $4-7$). La liaison de tous les sous-problèmes forme l'ensemble des clauses dures Σ_1. L'ensemble des clauses souples Σ_2 est constitué des clauses unitaires ϵ_i^j (ligne 3). La solution pour l'instance Partial-MAX-SAT formée par le couple (Σ_1, Σ_2) est une interprétation où toutes les clauses dures sont satisfaites et un nombre maximal de clauses ϵ_i^j sont satisfaites. Ainsi, les clauses originales δ_i^j correspondant aux ϵ_i^j affectées à *vrai*.

3.3 Conclusion

Un consensus entre agents est un paradigme intuitivement naturel qui peut jouer un rôle multiple dans différents domaines en intelligence artificielle. Nous avons vu à travers ce chapitre que différentes formes de définitions de notions de consensus peuvent être formulées. Récemment, (Grégoire *et al.* 2016c) ont introduit une approche définissant le concept de consensus dans le cadre de la logique propositionnelle. Dans cette approche, plusieurs critères de préférence pour le calcul de consensus sont à l'œuvre. Le critère de maximalité ($\subseteq$ et $\#$) en est un exemple; nous proposons une nouvelle technique de recherche d'un consensus maximal par rapport à l'inclusion ensembliste que nous décrirons dans la deuxième partie

de ce mémoire. Aussi, nous présenterons une nouvelle approche qui raffine le concept de consensus en concept de consensus admissible.

Contributions

Chapitre

4

Énumération de MCSes avec rotation de modèle

Sommaire

Extraire un sous-ensemble maximal satisfaisable de clauses (MSS) dans une formule CNF insatisfaisable est une tâche fondamentale dans de nombreux domaines de l'intelligence artificielle, allant du diagnostic d'erreurs (voir, par exemple, le travail précurseur de (Reiter 1987), le livre de (Hamscher *et al.* 1992) ou des travaux de recherche plus récents dans (Felfernig *et al.* 2012, Marques-Silva *et al.* 2015)), jusqu'aux différent paradigmes de changements de croyances (voir, par exemple, (Fermé et Hansson 2011)). Dans le cadre booléen, raisonner de manière crédule (Reiter 1980) sur des informations contradictoires représentées par une formule CNF Σ revient à raisonner sur un MSS de Σ. Le calcul d'un MSS (ou son complémentaire MCS) est, dans le pire cas, une tâche difficile du fait que le problème de décision correspondant consiste à vérifier si un ensemble de clauses est un MCS appartient à la classe de complexité *DP-complet* (Chen et Toda 1995). L'énumération des MCSes ou des MSSes d'une formule CNF Σ est une tâche encore plus difficile. Étant donné le nombre important de tels objets, cette tâche est une étape utile et parfois nécessaire pour mettre en œuvre certaines formes de raisonnement sceptique en argumentation (Lagniez *et al.* 2015b) ou encore dans les logiques non monotones (Bobrow 1980). L'énumération des MCSes peut aussi servir à l'analyse d'inconsistance d'un ensemble de clauses, tout comme le calcul de tous les sous-ensembles minimaux insatisfaisables (MUSes). Bien que le nombre de MCSes possibles d'une formule soit exponentiel dans le pire des cas, ce nombre reste petit dans de nombreuses situations pratiques. Cela rend la tentative d'énumération des MCSes réalisable dans ces cas.

Dans ce chapitre, nous présentons une technique qui améliore les approches les plus efficaces actuellement pour énumérer les MCSes d'une formule booléenne insatisfaisable Σ. Elle est basée sur le paradigme de *rotation de modèle* (Belov et Marques-Silva 2011, Nadel *et al.* 2014, Bacchus et Katsirelos 2015, Narodytska *et al.* 2018). Nous montrons que le processus de rotation de modèle permet de construire des ensembles de MCSes de Σ de manière heuristique et efficace. Ces travaux ont donné lieu à une publication internationale (Grégoire *et al.* 2018a).

Ce chapitre est organisé comme suit. La section 4.1 présente d'abord les propriétés utiles pour exploiter les clauses de transition dans le calcul des MCSes pour ensuite décrire notre nouvel algorithme pour l'énumération des MCSes. Dans le section 4.2, l'utilisation de la rotation du modèle dans ce dernier algorithme est expliquée. Ensuite, nous présentons les expérimentations qui nous ont permis d'évaluer notre nouvel outil d'énumération. Enfin, les pistes de recherche prometteuses sont brièvement présentées dans la conclusion.

4.1 Détecter plus de MCSes grâce aux clauses de transition

Les approches actuelles de l'état de l'art pour la recherche d'un MCS se basent généralement sur le concept de clause de transition (voir section 2.2). Dans ce travail, nous cherchons à exploiter les clauses de transition pour détecter plusieurs MCSes. À cet effet, nous avons défini des propriétés qui nous ont permis d'élaborer une méthode originale pour calculer récursivement un ensemble de MCSes. Naturellement, cette méthode s'intègre facilement dans l'approche directe pour l'énumération des MCSes.

À partir de maintenant, nous considérons Σ une formule CNF insatisfaisable formée par le couple $\langle \Sigma_1, \Sigma_2 \rangle$, où Σ_1 et Σ_2 sont respectivement l'ensemble des clauses dures et l'ensemble des clauses souples. Σ^S est la formule CNF obtenue après l'ajout des sélecteurs à Σ. Soulignons que les clauses dures n'ont pas besoin de sélecteurs, donc seulement les clauses de Σ_2 sont augmentées par des variables sélecteurs pour donner lieu à l'ensemble Σ_2^S. Dans le reste de ce chapitre, nous faisons souvent référence à Σ^S de manière implicite et nous ne faisons pas de différence entre les sélecteurs et les autres littéraux de Σ^S.

La propriété suivante montre que toute clause de transition α d'une sous-formule insatisfaisable $\Sigma' \subseteq \Sigma$ peut être le point de départ d'une famille de MCSes de la formule CNF Σ, chaque membre de cette famille contient la clause α. Nous montrons que cette famille capture les MCSes de Σ qui sont construits à partir de α et étendus avec tous les Partial-MCSes de $\langle \Sigma' \setminus \{\alpha\}, \Sigma \setminus \Sigma' \rangle$.

Propriété 4.1. Soient Σ une formule CNF insatisfaisable et la sous-formule $\Sigma' \subseteq \Sigma$ telle que Σ' contient au moins une clause de transition α. Pour tout Partial-MCS Γ pouvant être construit à partir de $\langle \Sigma' \setminus \{\alpha\}, \Sigma \setminus \Sigma' \rangle$, $\Gamma \cup \{\alpha\}$ est un MCS de Σ.

Preuve 4.1. Par l'absurde. Supposons que $\Gamma \cup \{\alpha\}$ n'est pas un MCS de Σ. Ceci dit soit (1) $\Sigma \setminus (\Gamma \cup \{\alpha\})$ est insatisfaisable ou bien (2) $\exists \beta \in \Gamma \cup \{\alpha\}$ tel que $\Sigma \setminus ((\Gamma \cup \{\alpha\}) \setminus \{\beta\})$ est satisfaisable.

Supposons que (1) est vrai. Étant donné que Γ est défini comme étant un Partial-MCS de $\langle \Sigma' \setminus \{\alpha\}, \Sigma \setminus \Sigma' \rangle$, nous avons que Γ est un MCS de $(\Sigma \setminus \{\alpha\})$. Cela implique que $((\Sigma \setminus \{\alpha\}) \cup \{\alpha\}) \setminus (\Gamma \cup \{\alpha\})$ est satisfaisable. Cette dernière formule peut être simplifiée à $\Sigma \setminus (\Gamma \cup \{\alpha\})$, qui est donc aussi satisfaisable. Cela contredit (1). Par conséquent, (1) ne se produit jamais.

Supposons cette fois que (2) est vrai. Admettons que $\beta \in \Gamma$ (alors $\beta \neq \alpha$). Puisque Γ est un Partial-MCS de $\langle \Sigma' \setminus \{\alpha\}, \Sigma \setminus \Sigma' \rangle$, alors nous avons que Γ est un MCS de $\Sigma \setminus \{\alpha\}$. Ainsi, $(\Sigma \setminus \{\alpha\}) \setminus (\Gamma \setminus \{\beta\})$ est insatisfaisable. Cela implique que $((\Sigma \setminus \{\alpha\}) \cup \{\alpha\}) \setminus ((\Gamma \setminus \{\beta\}) \cup \{\alpha\})$ est insatisfaisable. Du moment que nous avons supposé que $\beta \neq \alpha$, nous avons $(\Gamma \setminus \{\beta\}) \cup \{\alpha\} = (\Gamma \cup \{\alpha\}) \setminus \{\beta\}$. Par conséquent, $\Sigma \setminus ((\Gamma \cup \{\alpha\}) \setminus \{\beta\})$ est insatisfaisable. Cela contredit notre hypothèse et donc β et α doivent obligatoirement représenter la même clause.

Étant donné que Γ est un Partial-MCS de $\langle \Sigma' \setminus \{\alpha\}, \Sigma \setminus \Sigma' \rangle$, nous avons $(\Sigma' \setminus \{\alpha\}) \cap \Gamma = \emptyset$ et $\Gamma \subseteq \Sigma \setminus \Sigma'$. Par hypothèse $\alpha \in \Sigma'$, cela signifie que $\alpha \notin \Gamma$ et donc $((\Sigma' \setminus \{\alpha\}) \cup \{\alpha\}) \cap \Gamma = \emptyset$. Donc, $\Sigma' \cap \Gamma = \emptyset$. Si $\beta = \alpha$, nous avons que $\Sigma \setminus ((\Gamma \cup \{\alpha\}) \setminus \{\alpha\}) = \Sigma \setminus \Gamma$ est satisfaisable, d'après l'hypothèse (2). Comme $\Sigma' \subseteq \Sigma$ nous avons aussi $\Sigma' \setminus \Gamma$ qui est satisfaisable et puisque $\Sigma' \cap \Gamma = \emptyset$ nous avons que Σ' est satisfaisable. Cela contredit l'hypothèse qui assure que Σ' est un sous-ensemble insatisfaisable de Σ.

La propriété 4.1 est facilement adaptée pour traiter le cas où la recherche des Partial-MCSes est l'objectif visé, comme le montre le corollaire suivant.

Corollaire 4.2. Soit $\langle \Sigma_1, \Sigma_2 \rangle$ un couple de formules CNF tel que Σ_1 est satisfaisable et $\Sigma_1 \cup \Sigma_2$ est insatisfaisable. Considérons la sous-formule CNF $\Sigma_2' \subseteq \Sigma_2$ telle que $\Sigma_1 \cup \Sigma_2'$ contient au moins une clause de transition α de Σ_2'. Pour tous les Partial-MCSes Γ qui peuvent être construits à partir de $\langle \Sigma_1 \cup \Sigma_2' \setminus \{\alpha\}, \Sigma_2 \setminus \Sigma_2' \rangle$, nous avons $\Gamma \cup \{\alpha\}$ qui est un Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$.

Preuve 4.2. La propriété 4.1 nous permet de conclure que $\Gamma \cup \{\alpha\}$ est un MCS de $\Sigma_1 \cup (\Sigma'_2 \setminus \{\alpha\}) \cup (\Sigma_2 \setminus \Sigma'_2) \cup \{\alpha\}$, et donc de $\Sigma_1 \cup \Sigma_2$. Comme $\Gamma \cup \{\alpha\} \subseteq \Sigma_2$, nous concluons de manière directe que $\Gamma \cup \{\alpha\}$ est un Partial-MCS de $\langle \Sigma_1, \Sigma_2 \rangle$.

Admettons maintenant que nous calculons plusieurs clauses de transition pour une sous-formule CNF donnée $\Sigma' \subseteq \Sigma$. Pour chaque clause de transition, il est possible d'appliquer de manière récursive le corollaire précédent afin de construire plusieurs Partial-MCSes. De plus, la propriété suivante garantit que ces Partial-MCSes sont tous différents.

Propriété 4.3. Soit $\langle \Sigma_1, \Sigma_2 \rangle$ un couple de formules CNF telle que Σ_1 est satisfaisable et $\Sigma_1 \cup \Sigma_2$ est insatisfaisable. Considérons $\Sigma'_2 \subseteq \Sigma_2$ telle que $\Sigma_1 \cup \Sigma'_2$ contient au moins deux clauses de transition α_1 et α_2 de Σ'_2 ($\alpha_1 \in \Sigma'_2$ et $\alpha_2 \in \Sigma'_2$). Alors, pour tous les Partial-MCSes Γ_1 et Γ_2 qui peuvent être construits respectivement à partir de $\langle \Sigma_1 \cup \Sigma'_2 \setminus \{\alpha_1\}, \Sigma_2 \setminus \Sigma'_2 \rangle$ et $\langle \Sigma_1 \cup \Sigma'_2 \setminus \{\alpha_2\}, \Sigma_2 \setminus \Sigma'_2 \rangle$, nous avons que $\Gamma_1 \cup \{\alpha_1\}$ et $\Gamma_2 \cup \{\alpha_2\}$ sont des Partial-MCSes différents de $\langle \Sigma_1, \Sigma_2 \rangle$.

Preuve 4.3. Du corollaire 4.2 il est facile de montrer que $\Gamma_1 \cup \{\alpha_1\}$ et $\Gamma_2 \cup \{\alpha_2\}$ sont tous deux des Partial-MCSes de $\langle \Sigma_1, \Sigma_2 \rangle$. À présent, montrons que ces deux Partial-MCSes sont différents. Pour cela il suffit de montrer que $\alpha_1 \notin \Gamma_2$. Par définition d'un Partial-MCS, nous avons $\Gamma_2 \subseteq \Sigma_2 \setminus \Sigma'_2$. Comme $\alpha_1 \in \Sigma'_2$, il est évident que $\alpha_1 \notin \Gamma_2$ et donc $\Gamma_1 \cup \{\alpha_1\} \neq \Gamma_2 \cup \{\alpha_2\}$.

Cette dernière propriété (4.3) nous permet de dériver et de justifier un algorithme original et récursif décrit dans Algorithme 4.1, appelé `TC-MCS`, pour *Transition-Clauses-Based Enumeration of* MCS*es*. Cet algorithme original est une extension de l'algorithme `ELS` pour calculer cette fois non pas un mais plusieurs MCSes de manière récursive grâce à l'utilisation de la méthode de rotation de modèle (Algorithme 4.1 : lignes $9-14$). Il prend en entrée un couple de formules CNF $\langle \Sigma_1 \cup \Sigma_2^S, U \rangle$. Il renvoie en sortie un ensemble de Partial-MCSes de ce couple. L'ensemble U contient les sélecteurs correspondant aux clauses qui n'ont pas encore été insérées dans le MSS ou bien le MCS en construction. On considère que Σ_1 est satisfaisable et donc, que $\Sigma_1 \cup \Sigma_2^S$ est satisfaisable. Σ_2^S est en effet obtenu à partir de Σ_2 en rajoutant des sélecteurs aux clauses de Σ_2. Rappelons que les clauses dures n'ont pas besoin de sélecteurs puisqu'elles doivent toujours être satisfaites et donc être toujours activées. Les Partial-MCSes de $\langle \Sigma_1, \Sigma_2 \rangle$ sont obtenus directement à partir des Partial-MCSes de $\langle \Sigma_1 \cup \Sigma_2^S, S \rangle$, où S est l'ensemble des clauses unitaires s_i des sélecteurs de la formule augmentée Σ_2^S.

L'algorithme commence par vérifier si U est un ensemble vide. Dans le cas positif, la procédure retourne $\emptyset$ puisque l'ensemble des Partial-MCSes de $\langle \Sigma_1 \cup \Sigma_2^S, U \rangle$ est vide du fait que $\Sigma_1 \cup \Sigma_2^S$ est satisfaisable par construction. Cette première étape de l'algorithme est non-récursive. Dans le cas négatif, l'ensemble des Partial-MCSes Θ calculés et l'ensemble des sélecteurs M^+ sont initialisés à l'ensemble vide. Rappelons que M^+ sauvegarde les sélecteurs des clauses à activer lorsque $\Sigma_1 \cup \Sigma_2^S \cup M^+$ est satisfaisable. s_α est initialisé à $\top$ (symbole de tautologie).

Ensuite, dans la boucle *While* (lignes $5-8$), M^+ est augmenté incrémentalement par une séquence de sélecteurs s_α, qui sont retirés de U. Ce processus est itéré tant que U n'est pas vide et que s_α n'est pas une clause de transition de $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$. s_α est une clause de transition lorsque $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$ devient insatisfaisable. En effet, tous les sélecteurs insérés jusqu'ici dans M^+ offrent la garantie que $\Sigma_1 \cup \Sigma_2^S \cup M^+$ demeure satisfaisable. Ainsi, lorsque le sélecteur considéré s_α rend la formule insatisfaisable, celui-ci est alors identifié comme étant une clause de transition. À la fin de la boucle, deux cas peuvent se produire : (1) $s_\alpha \in M^+$, dans ce cas de figure, M^+ a capturé tous les sélecteurs de U et la formule d'entrée $\Sigma_1 \cup \Sigma_2^S \cup U$ était en effet satisfaisable. Par conséquent, un ensemble vide est renvoyé en sortie ; (2) $s_\alpha \notin M^+$ et ainsi s_α est une clause de transition. Dans ce cas, le CORE δ de la formule courante insatisfaisable $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$ calculé par le solveur SAT est récupéré

Algorithme 4.1 : `TC-MCS`$(\langle \Sigma_1 \cup \Sigma_2^S, U \rangle)$

Input : $\langle \Sigma_1 \cup \Sigma_2^S, U \rangle$ un couple de formules CNF
Output : Θ un ensemble de Partial-MCSes de $\langle \Sigma_1 \cup \Sigma_2^S, U \rangle$

1 **if** $U = \emptyset$ **then return** $\emptyset$;
2 $\Theta \leftarrow \emptyset$;
3 $M^+ \leftarrow \emptyset$;
4 $s_\alpha \leftarrow \top$;
5 **while** $U \neq \emptyset$ **and** $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$ *est satisfaisable* **do**
6 $\quad M^+ \leftarrow M^+ \cup \{s_\alpha\}$;
7 $\quad s_\alpha \leftarrow$ **choose** $s_\alpha \in U$;
8 $\quad U \leftarrow U \setminus \{s_\alpha\}$;
9 **if** $s_\alpha \notin M^+$ **then**
10 $\quad \delta \leftarrow$ `Core`$(\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\})$;
11 $\quad T \leftarrow \{s_\alpha\} \cup$ `Find-TC`(δ);
12 $\quad$ **foreach** $s_\beta \in T \cap M^+$ **do**
13 $\quad\quad \Omega \leftarrow$ `TC-MCS`$(\langle \Sigma_1 \cup \Sigma_2^S \cup (\delta \setminus \{\neg s_\beta\}), U \cup (M^+ \setminus \delta) \rangle)$;
14 $\quad\quad \Theta \leftarrow \Theta \cup (\beta \times \Omega)$;
15 **return** Θ;

(ligne 10). Évidement le sélecteur s_α est inclus dans le CORE δ et il est une clause de transition de δ. Comme nous l'avons déjà mentionné, le CORE renvoyé par un solveur SAT moderne est souvent plus petit que la formule prouvée inconsistante. Nous préférons ainsi utiliser ce CORE au lieu de considérer la formule insatisfaisable en entier, car le CORE contient généralement plus de clauses de transition. Il est facile de prouver par contradiction que chaque clause de transition est présente dans un CORE. Une fois le CORE de $(\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\})$ récupéré, une procédure `Find-TC` est appelée afin d'identifier les clauses de transition de ce dernier. Dans la prochaine section, nous présentons une approche pour réaliser ce processus, mais pour le moment, nous supposons que les clauses de transition sont extraites par une procédure appelée `Find-TC`. Notons que nous n'avons aucune garantie que cette approche soit efficace et qu'elle détecte la clause de transition s_α ; c'est la raison pour laquelle nous insérons directement s_α dans T (T est l'ensemble des clauses de transition) dans le but d'assurer la terminaison de l'algorithme. Ensuite, pour chaque clause de transition $s_\beta \in (T \cap M^+)$ identifiée, un appel récursif est effectué avec $\langle \Sigma_1 \cup \Sigma_2^S \cup (\Gamma \setminus \{\neg s_\beta\}), U \cup (M^+ \setminus \Gamma) \rangle$ comme entrée. Il est facile de montrer que ces paramètres correspondent aux conditions énoncées dans le corollaire 4.2. Ainsi, tous les Partial-MCSes pouvant être calculés à partir de $\langle \Sigma_1 \cup \Sigma_2^S \cup (\Gamma \setminus \{\neg s_\beta\}), U \cup (M^+ \setminus \Gamma) \rangle$ sont étendus par s_β pour produire des Partial-MCSes de la formule fournie en entrée.

`TC-MCS`$(\langle \Sigma_1 \cup \Sigma_2^S, U \rangle)$ peut être inséré dans l'algorithme 2.7 pour l'énumération des MCSes. Le nouvel algorithme d'énumération de MCSes est décrit dans l'algorithme 4.2.

Dans la section suivante, nous expliquons comment on peut utiliser la rotation de modèle pour rechercher des clauses de transition supplémentaires dans un CORE.

Algorithme 4.2 : `Enum-ELS-RMR`

Input : une formule CNF insatisfaisable Σ
Output : tous les MCSes de Σ

1 $\Sigma^S \leftarrow \{\alpha \vee \neg s_\alpha | \alpha \in \Sigma\}$; // avec s_α de nouvelles variables
2 $S \leftarrow \{s_\alpha | \alpha \in \Sigma\}$; // un ensemble de sélecteurs
3 $\Delta \leftarrow \emptyset$;
4 **while** $\Sigma^S \cup \Delta$ *est satisfaisable* **do**
5 $\quad \Theta \leftarrow$ TC-MCS $(\langle \Sigma^S \cup \Delta, S \rangle)$;
6 $\quad$ **foreach** $M^- \in \Theta$ **do**
7 $\quad\quad output(M^-)$;
8 $\quad\quad \Delta \leftarrow \Delta \cup (\bigvee_{s_\alpha \in M^-} s_\alpha)$; // clauses bloquantes

4.2 Utilisation de la rotation de modèle

La procédure `Find-TC` (ligne 11 de l'algorithme 4.1) implémente une méthode permettant de trouver de nouvelles clauses de transition. Une méthode naïve pour la détection de clauses de transition supplémentaires consisterait à calculer tous les MCSes singletons, c'est-à-dire de taille un, de la formule $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$ inclus dans M^+. Toutefois, une telle méthode directe et complète peut être très coûteuse en terme de temps de calcul, alors pour éviter ce problème nous proposons d'utiliser une technique incomplète pour rechercher des clauses de transition supplémentaires, basée sur le paradigme de rotation de modèle récursive noté `rmr` (pour *recursive model rotation*) (Belov et Marques-Silva 2011).

Ce paradigme a été initialement défini dans le contexte du calcul d'un ou plusieurs ensembles minimaux insatisfaisables (MUSes) d'une formule CNF insatisfaisable, où la recherche des clauses de transition supplémentaires de manière rapide peut également avoir un rôle déterminant dans l'efficacité de la détection des MUSes. Pareillement que (Bacchus et Katsirelos 2015) où l'objectif est l'énumération des MUSes, nous tirons avantage de la dualité entre MUSes et MCSes. Sauf que, dans notre cas nous appliquons le paradigme de `rmr` pour une tâche différente, qui consiste à énumérer des MCSes. `rmr` repose sur l'idée suivante : une clause de transition d'une formule CNF Σ est n'importe quelle clause $\alpha \in \Sigma$ tel qu'il existe une interprétation complète μ de Σ qui satisfait $\Sigma \setminus \{\alpha\}$ (et falsifie α, sinon la formule Σ serait satisfaisable). En partant d'un modèle μ de $\Sigma \setminus \alpha$, `rmr` consiste à « flipper » les valeurs de vérité des variables de la clause α dans μ et vérifier si la nouvelle interprétation μ' satisfait toutes les clauses de Σ hormis une certaine clause α' de Σ. Dans le cas positif, α' est marquée comme étant une clause de transition de Σ, à condition qu'elle ne soit pas déjà marquée comme telle, et le processus est répété récursivement avec μ' et α'. Le processus `rmr` est dépeint dans l'algorithme 4.3.

`rmr` peut ainsi calculer plusieurs clauses de transition une fois qu'une première clause de transition a été identifiée dans la formule. Dans la recherche d'un Partial-MCS, cette situation se produit lorsque nous avons montré qu'il existe un modèle μ de $\Sigma_1 \cup \Sigma_2^S \cup M^+$ et prouvé que $\Sigma_1 \cup \Sigma_2^S \cup M^+ \cup \{s_\alpha\}$ est insatisfaisable. μ représente alors l'interprétation initiale à fournir au processus `rmr`. Les clauses de transition à garder sont uniquement celles appartenant à $\{\beta \in \Sigma_2 \mid s_\beta \in M^+\}$. Il est important de noter que nous nous assurons que les variables sélecteurs ne sont jamais flippées par le processus `rmr`.

Bien que le processus `rmr` soit polynomial, il s'avère qu'en pratique cela peut prendre beaucoup de temps de calcul ; cela peut donc réduire l'efficacité pratique de l'algorithme d'énumération. Une telle situation se produit lorsque la taille de l'ensemble des clauses bloquantes Δ devient trop grand. Pour éviter un tel inconvénient, nous montrons que certaines conditions à réunir suffisent pour ne pas prendre en considération les clauses de Δ dans le processus `rmr`.

Algorithme 4.3 : `rmr` (*Recursive Model Rotation*)

Input : une formule CNF Σ ;
T un ensemble de clauses de transition de Σ ;
α une clause de transition de Σ ;
μ un modèle de $(\Sigma \setminus \{\alpha\})$;

```
1 foreach x ∈ Var(α) do
2     μ' ← μ|_¬x;                                  // flipper la variable x dans μ
3     if falsifie(μ', Σ) = {α'} and α' ∉ T then
4         T ← T ∪ {α'};
5         rmr(Σ, T, α', μ');
```

D'abord, démontrons l'importance que peut avoir Δ dans la recherche des clauses de transition supplémentaires. Considérons $\langle \Sigma^S, S \rangle$ avec $\Sigma^S = \{\neg s_1 \vee a \vee b, \neg s_2 \vee \neg a, \neg s_3 \vee \neg b\}$. Supposons qu'un MCS, à savoir $\{s_1\}$, a déjà été calculé. Ainsi, à cette étape $\Delta = \{s_1\}$. Nous allons maintenant itérer le processus et calculer un MCS supplémentaire et appeler `TC-MCS` sur $\langle \Sigma^S \cup \Delta, S \rangle$. Admettons que nous arrêtions la boucle principale de `TC-MCS` lorsque $M^+ = \{s_1, s_2\}$ et $s_\alpha = s_3$. La condition à la ligne 9 (Algorithme 4.1) est satisfaite. Un CORE qui est composé de toute la formule est calculé (ligne 10). Ensuite, nous nous mettons à chercher d'autres clauses de transition dans $\Sigma^S \cup S$ au lieu de $\Sigma^S \cup S \cup \Delta$. Dans ce cas, il est facile de montrer que s_1, s_2, s_3 sont des clauses de transition et un des MCSes sera calculé deux fois. Évidemment, vérifier a posteriori si un MCS a déjà été trouvé en consultant Δ peut être coûteux en temps de calcul, c'est pour cela que nous voulons éviter ce processus.

Nous savons que Σ^S et Δ ne partagent pas de littéraux. En effet, Δ est un ensemble de clauses positives composées de variables sélecteurs, tandis que ces variables sélecteurs apparaissent négativement dans Σ^S. Par conséquent, la satisfaction de $\Sigma^S \cup \Delta$ peut être divisée en deux sous-problèmes indépendants suivant une partition $\{P, N\}$ de l'ensemble des sélecteurs, où P et N désignent respectivement l'ensemble des sélecteurs positifs et l'ensemble des négatifs. Concrètement :

Propriété 4.4. Soient $\{P, N\}$ une partition de S, Σ^S une formule CNF augmentée avec un ensemble de sélecteurs S et Δ une formule CNF composée uniquement de variables sélecteurs telle que toutes ses clauses sont positives. $\Sigma^S \cup \Delta \cup \bigwedge_{s \in P} s \cup \bigwedge_{s \in N} \neg s$ est satisfaisable si et seulement si $\Sigma^S \cup \bigwedge_{s \in P} s$ et $\Delta \cup \bigwedge_{s \in N} \neg s$ est satisfaisable.

Preuve 4.4. Démontrons que si $\Sigma^S \cup \Delta \cup \bigwedge_{s \in P} s \cup \bigwedge_{s \in N} \neg s$ alors $\Sigma^S \cup \bigwedge_{s \in P} s$ et $\Delta \cup \bigwedge_{s \in N} \neg s$ sont satisfaisables et *vice versa*.
($\Rightarrow$) direct.
($\Leftarrow$) si $\Sigma^S \cup \bigwedge_{s \in P} s$ et $\Delta \cup \bigwedge_{s \in N} \neg s$ sont toutes les deux satisfaisables, alors il existe un modèle μ qui satisfait $\Sigma^S \cup \bigwedge_{s \in P} s$. Par construction de Σ^S, quelle que soit l'interprétation considérée, il est toujours possible de flipper la valeur de vérité des sélecteurs de 1 (*vrai*) à 0 (*faux*) et garder l'interprétation résultante μ_P telle que μ_P est un modèle de Σ^S (ceci est possible car l'affectation des sélecteurs à 0 désactive des clauses de Σ^S et donc réduit cette formule). Donc, puisque $P \cap N = \emptyset$, on peut construire une interprétation μ_P^N qui est équivalente à μ_P excepté sur les valeurs de vérité des sélecteurs appartenant à N où on impose leur affectation à 0. Il est clair que μ_P^N est aussi un modèle de $\Delta \cup \bigwedge_{s \in N} \neg s$. Par construction, Δ contient uniquement des clauses positives composées de variables sélecteurs. Donc, quel que soit le modèle de $\Delta \cup \bigwedge_{s \in N} \neg s$ considéré, il est toujours possible de flipper la valeur de vérité de certains sélecteurs de 0 à 1 et de garder cette interprétation comme modèle de Δ. Donc, puisque tous les modèles de $\Delta \cup \bigwedge_{s \in N} \neg s$ doivent satisfaire $\bigwedge_{s \in N} \neg s$, on peut construire l'interprétation $\bigwedge_{s \in N} \neg s \wedge$

$\bigwedge_{s\in P} s$ qui satisfait $\Delta \cup \bigwedge_{s\in N} \neg s$. Donc, comme Δ est formée uniquement de variables sélecteurs, il est facile de montrer que μ_P^N satisfait également $\Delta \cup \bigwedge_{s\in N} \neg s$. En conséquence, μ_P^N satisfait $\Sigma^S \cup \Delta \cup \bigwedge_{s\in P} s \cup \bigwedge_{s\in N} \neg s$.

La construction d'un MCS peut se voir de manière implicite comme un calcul de bi-partition de l'ensemble des sélecteurs S en $\{M^+, M^-\}$ tel que $\Sigma^S \cup \bigwedge_{s\in M^+} s$ et $\Delta \cup \bigwedge_{s\in M^-} \neg s$ soient satisfaisables. Le principe est de déplacer autant que possible des sélecteurs de M^- à M^+ tout en maintenant la satisfaisabilité dans $\Sigma^S \cup \bigwedge_{s\in M^+} s$ et $\Delta \cup \bigwedge_{s\in M^-} \neg s$. Il est facile de prouver que si la bi-partition $\{M^+, M^-\}$ de S est de manière à ce que $\Sigma^S \cup \bigwedge_{s\in M^+} s$ et $\Delta \cup \bigwedge_{s\in M^-} \neg s$ sont satisfaisables, alors si on déplace un élément s' de M^- à M^+ tel que $\Sigma^S \cup \bigwedge_{s\in M^+\cup\{s'\}} s$ est satisfaisable, alors $\Delta \cup \bigwedge_{s\in M^-\setminus\{s'\}} \neg s$ l'est aussi (comme toutes les clauses de Δ sont positives, l'affectation d'un sélecteur positif ne peut pas rendre la formule Δ insatisfaisable). Reste maintenant à prouver l'autre sens de l'implication, c'est-à-dire, lorsqu'un élément de M^+ est déplacé vers M^-.

Afin d'éviter de considérer Δ dans la procédure `rmr`, nous proposons le processus suivant. D'abord, calculer une bi-partition $\{M^+, M^-\}$ de S qui assure que $\Sigma^S \cup \bigwedge_{s\in M^+} s$ et $\Delta \cup \bigwedge_{s\in M^-} \neg s$ sont satisfaisables. M^+ est utilisé comme ensemble initial à étendre pour avoir un MSS et toutes les clauses de M^- sont marquées comme étant candidates pour être identifiées comme des clauses de transition. Ensuite, à chaque fois qu'un nouveau sélecteur est ajouté dans M^+, on a $\Sigma^S \cup \bigwedge_{s\in M^+} s$ qui est satisfaisable. Notons qu'ajouter un élément dans M^+ c'est dans un certain sens "déplacer" cet élément de M^- à M^+. Par conséquent, $\Sigma^S \cup \bigwedge_{s\in M^+} s$ et $\Delta \cup \bigwedge_{s\in M^-} \neg s$ sont tous les deux satisfaisables. D'une certaine manière, les sélecteurs qui ont été marqués sont responsables de la satisfaisablité de Δ. Ainsi, puisque M^+ est construit de manière à ce que $\Sigma^S \cup \bigwedge_{s\in M^+} s$ soit satisfaisable, il devient inutile de vérifier la satisfaisabilité de Δ pendant le déroulement du processus `rmr` quand on interdit que les clauses de Δ soient sélectionnées comme clauses de transition.

4.3 Expérimentations

Nous avons implanté tous nos algorithmes en language C++ et utilisé MINISAT [6] comme solveur SAT. Nous avons sélectionné un ensemble de 866 benchmarks utilisé dans (Previti *et al.* 2017, Marques-Silva *et al.* 2013), dont 269 instances sont des *plain* MAX-SAT [7] et les 597 instances restantes sont des Partial-MAX-SAT. Nous avons enrichi ce jeu d'instances en considérant aussi une deuxième série de benchmarks *plain* MAX-SAT issue des instances de la dernière compétition de calcul de MUS [8]. Certaines de ces instances sont déjà présentes dans le jeu d'instances proposé par (Previti *et al.* 2017). Comme nous avons gardé uniquement celles qui n'existaient pas au départ, 1090 benchmarks sont considérés au total : 493 instances *plain* MAX-SAT et 597 Partial-MAX-SAT.

Toutes les expérimentations ont été exécutées sur des machines équipées d'un processeur Intel Xeon E52643 cadencé à 3.30GHz, avec 64 Gb de mémoire vive et sous le système d'exploitation Linux CentOS. Le *time-out* a été réglé à 1800 secondes pour chaque exécution d'un algorithme sur une instance et le *memory-out* a été réglé à 8 Gb pour chaque exécution. L'ensemble des données, résultats et outils utilisés dans cette étude expérimentale sont disponibles à l'adresse `http://www.cril.fr/enumcs`.

Nous avons comparé les performances des différents algorithmes en termes de nombre de MCSes énumérés dans la limite du *time-out* par instance. Les résultats sont présentés en forme de *scatter plot*

6. `http://minisat.se/`
7. Une instance *plain* MAX-SAT est une instance du problème MAX-SAT où toutes les clauses de celle-ci sont des clauses souples, autrement dit, l'instance ne contient pas de clauses dures.
8. `http://www.satcompetition.org/2011`

(ou nuage de points), tel que pour chaque point la valeur en abscisse représente le nombre de MCSes calculés par l'algorithme affiché dans l'axe horizontal et la valeur en ordonnée représente le nombre de MCSes calculés par l'algorithme affiché dans l'axe vertical.

D'abord, nous avons comparé notre implantation de `Enum-ELS-RMR` (Algorithme 4.2) avec le même algorithme sans processus de rotation de modèle `rmr`, c'est-à-dire avec `Enum-ELS` l'algorithme 2.7 pour l'énumération des MCSes, utilisant `ELS` comme méthode d'extraction d'un MCS. Notre version de `ELS` inclut la technique exploitant les modèles retournés par le solveur SAT (Grégoire *et al.* 2014a) (voir section 7), ainsi que la technique des littéraux *backbones* de (Marques-Silva *et al.* 2013) (voir section 7).

Comme le montre la figure 4.1, le paradigme de `rmr` permet de calculer plus (ou le même nombre) de MCSes sur les instances MAX-SAT simples (plain MAX-SAT). Un même résultat a été obtenu pour la plupart des tests sur les instances Partial-MAX-SAT.

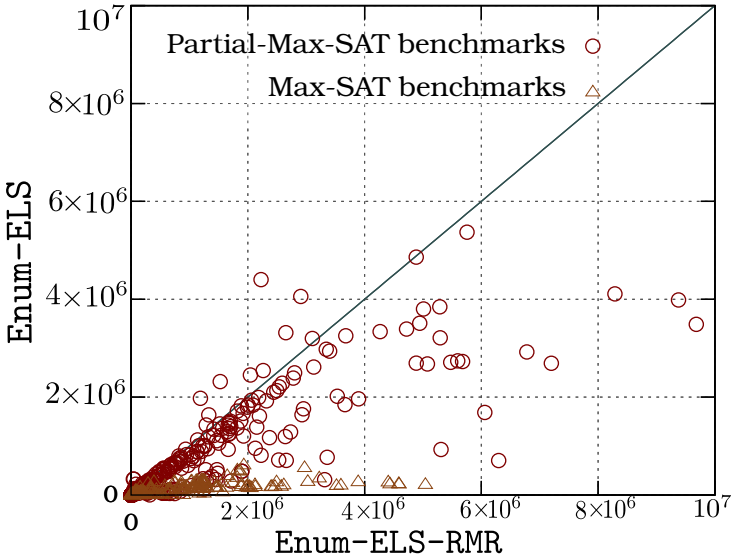

FIGURE 4.1 – `Enum-ELS-RMR` vs. `Enum-ELS`

Ensuite, nous avons combiné la technique de *caching* des CORES (Previti *et al.* 2017) avec `Enum-ELS-RMR`, donnant lieu au nouvel algorithme `Enum-ELS-RMR-Cache`. Comme suggéré dans (Previti *et al.* 2017), nous n'avons pas utilisé l'option des littéraux *backbones* dans `ELS` car cela pourrait ralentir la technique de mise en *cache*. `Enum-ELS-RMR-Cache` affiche un meilleur score en nombre de MCSes calculés sur une grande partie des benchmarks (Figure 4.2). Notons que nous avons constaté que sur les instances pour lesquelles `Enum-ELS-RMR-Cache` affiche un nombre réduit de MCSes énumérés, `Enum-ELS-Cache` (l'algorithme `Enum-ELS` utilisant le *caching*) aussi obtient des résultats moins bons comparé à `Enum-ELS-RMR`. De plus, l'intégration du *caching* des CORES tend à prendre plus d'espace mémoire et dépasser la limite du *memory-out*. Ces deux points peuvent être expliqués par le fait que, comme il est souligné déjà dans (Previti *et al.* 2017), l'espace mémoire nécessaire au *cache* peut devenir trop grand et donc difficile à gérer.

Enfin, nous avons comparé l'outil `mcscache-els` de (Previti *et al.* 2017), qui implante la technique de *caching* des CORES sur une version de l'état de l'art de `Enum-ELS`, avec `Enum-ELS-RMR-Cache`. La figure 4.3 montre clairement que `Enum-ELS-RMR-Cache` surpasse `mcscache-els` sur la majorité des instances et la différence entre les deux approches apparaît plus importante sur les instances dont le nombre de MCSes est très grand.

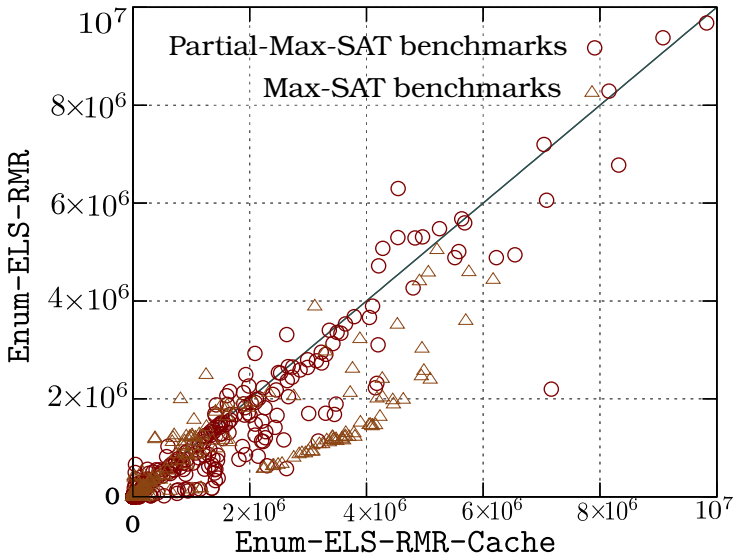

FIGURE 4.2 – `Enum-ELS-RMR-Cache` vs. `Enum-ELS-RMR`

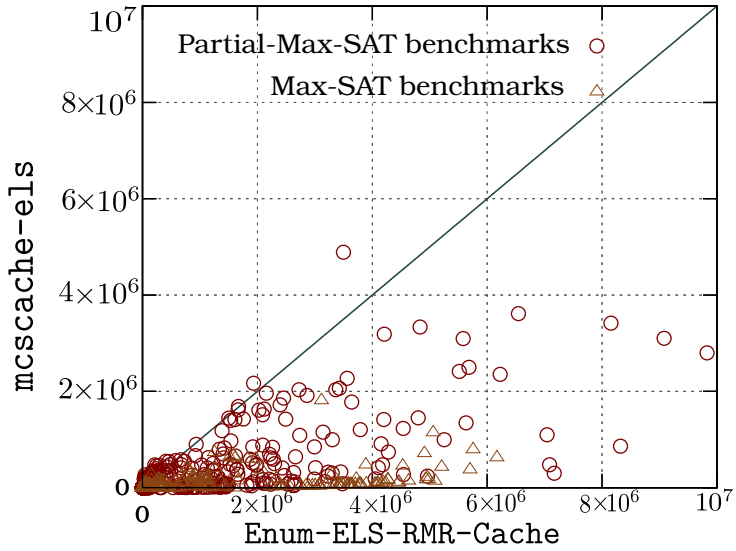

FIGURE 4.3 – `Enum-ELS-RMR-Cache` vs. `mcscache-els`

4.4 Conclusion

Dans cette contribution, nous avons introduit une technique qui améliore les performances des meilleures approches pour l'énumération des ensembles minimaux rectificatifs (MCSes) d'une formule CNF. Bien que le nombre de MCSes peut être exponentiel dans le pire des cas, dans de nombreux problèmes réels ce nombre reste relativement bas, en particulier dans les problèmes pour lesquels le nombre et la cardinalité des différentes sources minimales d'insatisfaisabilité restent faibles.

Les progrès réalisés dans l'énumération des MCSes d'une formule CNF lorsque le nombre des MCSes est grand, ouvrent de nouvelles perspectives pour l'énumération des MUSes (ensembles minimaux insatisfaisables). En effet, les approches d'énumération des MUSes basées sur la dualité entre MUSes et MCSes, plus précisément, celles qui calculent d'abord tous les MCSes avant de calculer MUSes, peuvent clairement tirer parti de ces améliorations obtenues dans l'énumération des MCSes.

Nous pensons que ces travaux ouvrent également d'autres voies pour d'autres recherches. Particulièrement, la propriété 4.4 ouvre la voie à une certaine parallélisation de la procédure d'énumération.

Enfin, comme le raisonnement sceptique en présence d'informations contradictoires peut équivaloir à calculer l'intersection de tous les sous-ensembles maximaux satisfaisables (MSSes), de nouveaux progrès dans l'énumération des MSSes, et donc des MCSes, peuvent s'avérer utiles pour la mise en œuvre de telles formes de raisonnement.

Chapitre

5 Extraction d'un AC-MSS

Sommaire

L'extraction efficace d'un sous-ensemble maximal d'informations cohérent avec des contextes multiples ou des sources d'informations supplémentaires est un problème clé dans de nombreux domaines de l'Intelligence Artificielle, en particulier, lorsque ces contextes ou sources peuvent être mutuellement contradictoires. Comme illustré dans (Besnard *et al.* 2015), ce problème est central en planification, dans la prise de décision, le diagnostic, l'argumentation et le raisonnement non monotone, ainsi que dans bien d'autres champs de l'Intelligence Artificielle.

Cette extraction pourrait être aussi à la base d'une méthode d'énumération des sous-ensembles maximaux cohérents avec toutes les hypothèses lorsque ces sous-ensembles demeurent en petit nombre et qu'un temps de calcul suffisant est disponible. Elle peut également être adaptée de façon à respecter un ordre de préférence entre informations.

Dans cette étude, cette question est étudiée d'un point de vue calculatoire pratique en logique propositionnelle, lorsque la maximalité se réfère à l'inclusion ensembliste. Une nouvelle approche est introduite et les expérimentations menées montrent qu'elle surpasse les méthodes existantes. Les travaux que nous présentons dans ce chapitre, ont donné lieu à deux publications, une internationale (Grégoire *et al.* 2016b) et l'autre francophone (Grégoire *et al.* 2016a).

À partir de maintenant, nous considérerons que Σ est un ensemble fini de clauses et que $\mathcal{AC}$ est un ensemble fini de contextes Γ_i, tels que chaque Γ_i est un ensemble satisfaisable de clauses.

Nous nous focaliserons sur l'extraction soit d'un AC-MSS$(\Sigma, \mathcal{AC})$, soit d'un AC-MCS$(\Sigma, \mathcal{AC})$, et ce de manière interchangeable, dans ce chapitre. Quand un des ensembles est retourné, l'autre se calcule de manière directe, puisque leur combinaison constitue une partition de Σ.

5.1 Algorithme incrémental basique

Nous avons présenté dans la section 2.4.2 les techniques d'extraction d'un AC-MSS. La méthode par transformation (Besnard *et al.* 2015) est la plus efficace dans la littérature. Cependant, le traitement d'un très grand nombre m de contextes dans $\mathcal{AC}$ ($m = card(\mathcal{AC})$) demeure problématique pour cette approche sur le plan pratique car la taille de l'instance transformée est proportionnelle à m.

Il existe une autre technique que la méthode par transformation (Besnard *et al.* 2015) et la méthode *brute-force* (Besnard *et al.* 2015) pour partitionner Σ en un AC-MSS$(\Sigma, \mathcal{AC})$ et un AC-MCS$(\Sigma, \mathcal{AC})$. En effet, une manière de calculer un AC-MSS/AC-MCS est de construire ce dernier incrémentalement.

Algorithme 5.1 : `Incremental`$_1$`-AC-MSS`

Input : Σ : un ensemble de clauses;
$\mathcal{AC} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: un ensemble d'ensembles satisfaisables de clauses;
Output : Φ t.q. Φ est un AC-MSS$(\Sigma, \mathcal{AC})$;

1 $\Phi \longleftarrow$ `Greedy-AC-SS`$(\Sigma, \mathcal{AC})$;
2 $\Psi \longleftarrow (\Sigma \setminus \Phi)$;
3 **foreach** $\alpha \in \Psi$ **do**
4 `all-`Γ_i`-satisfied` $\longleftarrow$ `true;`
5 $i \longleftarrow 1$;
6 **while** $i \leq m$ **and** `all-`Γ_i`-satisfied` **do**
7 `all-`Γ_i`-satisfied` $\longleftarrow$ $\Phi \cup \{\alpha\} \cup \Gamma_i$ `is satisfiable;`
8 $i \longleftarrow i + 1$;
9 **if** `all-`Γ_i`-satisfied` **then** $\Phi \longleftarrow \Phi \cup \{\alpha\}$;
10 **return** Φ;

`Incremental`$_1$`-AC-MSS` est décrit dans l'Algorithme 5.1. Initialement, il appelle la procédure `Greedy-AC-SS` qui renvoie Φ, le AC-MSS$(\Sigma, \mathcal{AC})$ en construction. Ensuite, Φ est augmenté par les clauses α de $\Sigma \setminus \Phi$ tel que, à chaque itération, $\Phi \cup \{\alpha\}$ est satisfaisable avec tous les Γ_i. La validité de cet algorithme se démontre facilement à partir des trois points de la Définition 2.11 : la satisfaisabilité de Φ est garantie à chaque étape ; par construction, l'ensemble final Φ ne contredit aucun Γ_i et pour chaque clause α de $\Sigma \setminus \Phi$ il existe au moins un Γ_i pour lequel $\Phi \cup \{\alpha\} \cup \Gamma_i$ est insatisfaisable.

Exemple 5.1. Soient $\Sigma = \{a \vee b, a \vee c, d \vee b, e \vee c, \neg b, \neg c\}$ et $\mathcal{AC} = \{\{\neg a\}, \{\neg d\}, \{\neg e\}\}$. Déroulons `Incremental`$_1$`-AC-MSS` sur cet exemple. Supposons que $\{a \vee b, a \vee c, e \vee c\}$ soit renvoyé par `Greedy-AC-SS`. Φ est initialisé à cet ensemble et ainsi $\Psi = \{d \vee b, \neg c, \neg b\}$ (ligne 1). Chaque clause de Ψ est alors considérée successivement. Quand $\alpha = d \vee b$, $\Psi \cup \{\alpha\}$ est satisfaisable avec chaque contexte et ainsi α est ajouté à Φ. Quand $\alpha = \neg c$ (et quand $\alpha = \neg b$), comme $\Phi \cup \{\alpha\}$ est insatisfaisable, $\neg a$ n'est pas ajouté à la solution en construction.

Indépendamment du coût de calcul de la procédure `Greedy-AC-SS`, `Incremental`$_1$`-AC-MSS` effectue mn appels à un solveur SAT dans le pire cas, où m et n représentent respectivement *card*$(\mathcal{AC})$ et *card*(Σ).

Nous avons utilisé une version plus efficace de `Greedy-AC-SS` (voir l'algorithme 5.2) dans nos expérimentations. Celle-ci consiste à partitionner progressivement Σ en un couple (Φ, Ψ), où Φ est le AC-SS$(\Sigma, \mathcal{AC})$ courant et Ψ est l'AC-CS$(\Sigma, \mathcal{AC})$ courant. Pour chaque Γ_i, un modèle μ de $\Gamma_i \cup \Phi$ est calculé. Les clauses de Ψ satisfaites par I sont sauvegardées dans Υ ; Φ et Ψ sont mis à jour, les clauses de Υ sont déplacées de Ψ vers Φ. La recherche des interprétations initiales est réalisée en appellant un solveur SAT sur Σ. Comme dans (Grégoire *et al.* 2014a), la technique dite "*progress saving* interpretation" est utilisée pour la recherche d'un modèle de $\Gamma_i \cup \Phi$ car elle permet de satisfaire un plus grand nombre de clauses de Σ.

Plus tard, nous parlerons des performances de l'algorithme `Incremental`$_1$`-AC-MSS` muni de ce pré-traitement. À présent, nous allons présenter notre nouvelle méthode incrémentale, plus élaborée et beaucoup plus performante que l'actuelle.

Algorithme 5.2 : `Greedy-AC-SS`

Input : Σ : un ensemble de clauses ;
$\mathcal{AC} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: un ensemble d'ensembles satisfaisables de clauses;
Output : $\Phi \subseteq \Sigma$ t.q. Φ est un AC-SS$(\Sigma, \mathcal{AC})$;

```
1  Ψ ⟵ Σ;  Φ ⟵ ∅;  cpt ⟵ 0;
2  repeat
3  |  ϒ ⟵ Ψ;
4  |  foreach Γ_i ∈ AC do
5  |  |  Soit μ un modèle de Γ_i ∪ Φ;
6  |  |  ϒ ⟵ ϒ ∩ {α ∈ Ψ | μ(α) = true};
7  |  Φ ⟵ Φ ∪ ϒ;
8  |  Ψ ⟵ Ψ \ ϒ;
9  |  cpt ⟵ cpt + 1;
10 until ϒ = ∅ or cpt > #iteration-max;
11 return Φ;
```

5.2 Propriétés utiles

Pour construire des algorithmes incrémentaux plus efficaces, nous nous sommes focalisés sur des propriétés qui nous permettent d'identifier les clauses à ajouter au AC-MSS$(\Sigma, \mathcal{AC})$ ou au AC-MCS$(\Sigma, \mathcal{AC})$, pendant leurs constructions. À partir de maintenant, nous travaillerons sur l'ensemble AC-MCS$(\Sigma, \mathcal{AC})$ qui forme le sujet de notre algorithme original.

Intuitivement, la première propriété se présente comme suit.

*Lorsqu'on sait que l'ajout d'une clause α à l'*AC-CS *en construction rendra le* AC-SS *cohérent avec tous les contextes et pas seulement avec une partie d'entre eux, alors il est approprié d'insérer α dans l'*AC-CS *courant. En effet, dans ces cas là, il existe forcément au moins un* AC-MCS *qui est inclus dans l'*AC-CS *courant contenant α. De plus, tous ces* AC-MCS *contiennent nécessairement α.* Formellement :

Propriété 5.1. Admettons que $\Theta \subset \Sigma$ et que $\alpha \in (\Sigma \setminus \Theta)$.

$$\text{Si } \exists\ \mathcal{AC}' \text{ tel que } \mathcal{AC}' \subseteq \mathcal{AC} \text{ et } \exists\ \Gamma_j \in \mathcal{AC}' \text{ tel que } \forall\ \Gamma_i \in \mathcal{AC} \setminus \mathcal{AC}'$$

1. $(\Sigma \setminus \Theta) \cup \Gamma_i$ est satisfaisable ;
2. $(\Sigma \setminus \Theta) \cup \Gamma_j$ est insatisfaisable ;
3. $\forall\ \Gamma_k \in \mathcal{AC}'$, $(\Sigma \setminus (\Theta \cup \{\alpha\})) \cup \Gamma_k$ est satisfaisable

alors il existe au moins un AC-MCS$(\Sigma, \mathcal{AC})$, nommé Ψ, tel que
(1) $\Psi \subseteq \Theta \cup \{\alpha\}$, et
(2) $\forall \Psi$ qui sont AC-MCS$(\Sigma, \mathcal{AC})$ tel que $\Psi \subseteq \Theta \cup \{\alpha\}$: $\alpha \in \Psi$.

Preuve 5.1. (1) Par hypothèse, on a $\forall \Gamma_i \in \mathcal{AC} \setminus \mathcal{AC}'$: $(\Sigma \setminus \Theta) \cup \Gamma_i$ est satisfaisable et $\forall \Gamma_k \in \mathcal{AC}'$: $(\Sigma \setminus (\Theta \cup \{\alpha\})) \cup \Gamma_k$ est satisfaisable. Donc, $\forall \Gamma_k \in \mathcal{AC}$: $(\Sigma \setminus (\Theta \cup \{\alpha\})) \cup \Gamma_k$ est satisfaisable puisque $\Sigma \setminus (\Theta \cup \{\alpha\}) \subset (\Sigma \setminus \Theta)$. Ainsi, $\Theta \cup \{\alpha\}$ est un AC-CS$(\Sigma, \mathcal{AC})$, puisqu'il est conforme à la définition d'un AC-CS de Σ sous les contextes $\mathcal{AC}$. Il est clair que tout AC-CS contient au moins un AC-MCS. Donc, $\exists \Psi \subseteq \Theta \cup \{\alpha\}$ tel que Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$.
(2) Par l'absurde. Admettons que $\exists\ \Psi$ tel que Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$, $\Psi \subseteq \Theta \cup \{\alpha\}$ et que $\alpha \notin \Psi$.

Donc, $\Psi \subseteq \Theta$ et d'après la condition 2, on a $(\Sigma \setminus \Psi) \cup \Gamma_j$ qui est satisfaisable, ce qui contredit la supposition que Ψ est AC-MCS$(\Sigma, \mathcal{AC})$.

Exemple 5.2. Considérons Σ et $\mathcal{AC}$ de l'exemple 5.1. Admettons que $\Theta = \{a \vee b, d \vee b, \neg b\}$, $\mathcal{AC}' = \{\{\neg a\}, \{\neg e\}\}$, $\alpha = \neg c$ et $\Gamma_j = \{\neg a\}$. Toutes les conditions sont satisfaites pour appliquer la Propriété 5.1. Ainsi, α appartient à tous les AC-MCS pouvant être obtenus à partir de Σ et $\mathcal{AC}$ et qui sont inclus dans $\{a \vee b, d \vee b, \neg b, \neg c\}$ (c'est-à-dire, $\{\{a \vee b, b \vee d, \neg c\}, \{\neg b, \neg c\}\}$).

Nous avons cherché à généraliser cette propriété dans le but de développer un algorithme qui extrait un AC-MCS$(\Sigma, \mathcal{AC})$.

La généralisation s'établit comme suit. On peut remplacer α dans la Propriété 5.1 par n'importe quel Partial-MCS de $\langle(\Sigma \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta\rangle$, où $\Sigma \setminus \Upsilon) \cup \Gamma_j$ et $\Upsilon \setminus \Theta$ représentent respectivement l'ensemble des clauses dures et l'ensemble des clauses souples. *Intuitivement, on peut étendre le* AC-CS *en construction, en lui ajoutant un ensemble minimal de clauses qui permet de rendre le* AC-SS *correspondant satisfaisable avec un context supplémentaire, à condition que le* AC-CS *étendu rende le* AC-SS *correspondant satisfaisable avec tous les contextes.*

Propriété 5.2. Supposons que Υ soit un AC-CS$(\Sigma, \mathcal{AC})$, $(\Theta \cup \Pi) \subseteq \Upsilon$ et $\Theta \cap \Pi = \emptyset$.

Si $\exists\ \mathcal{AC}'$ tel que $\mathcal{AC}' \subseteq \mathcal{AC}$ et $\exists\ \Gamma_j \in \mathcal{AC}'$ tel que $\forall\ \Gamma_i \in \mathcal{AC} \setminus \mathcal{AC}'$

1. $(\Sigma \setminus \Theta) \cup \Gamma_i$ est satisfaisable ;
2. $(\Sigma \setminus \Theta) \cup \Gamma_j$ est insatisfaisable ;
3. Π est un Partial-MCS de $\langle(\Sigma \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta\rangle$;
4. $\forall\ \Gamma_k \in \mathcal{AC}'$, $(\Sigma \setminus (\Theta \cup \Pi)) \cup \Gamma_k$ est satisfaisable.

donc il existe au moins un Ψ qui est un AC-MCS$(\Sigma, \mathcal{AC})$ tel que

(1) $\Psi \subseteq \Theta \cup \Pi$, et
(2) $\forall \Psi$ s.t. Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$ et $\Psi \subseteq \Theta \cup \Pi : \Pi \subseteq \Psi$.

Preuve 5.2. La preuve s'obtient directement en appliquant la propriété 5.1 où chaque $\beta \in \Pi$ peut prendre la place de α avec $(\Theta \cup (\Pi \setminus \{\beta\}))$ au lieu de Θ. D'abord, remarquons que, étant donné que Υ est un AC-CS$(\Sigma, \mathcal{AC})$, Partial-MCS de $\langle(\Sigma \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Phi\rangle$ est correctement défini. Ensuite, il est facile de vérifier que les trois points suivants sont satisfaits quelle que soit la valeur de β de Π :

- $(\Sigma \setminus (\Theta \cup (\Pi \setminus \{\beta\}))) \cup \Gamma_i$ est satisfaisable. En effet, comme $(\Sigma \setminus \Theta) \cup \Gamma_i$ est satisfaisable, $\forall \Omega \subseteq (\Sigma \setminus \Theta) : \Omega \cup \Gamma_i$ est aussi satisfaisable ;
- $(\Sigma \setminus (\Theta \cup (\Pi \setminus \{\beta\}))) \cup \Gamma_j$ est insatisfaisable. Si $(\Sigma \setminus (\Theta \cup (\Pi \setminus \{\beta\}))) \cup \Gamma_j$ est satisfaisable, alors supprimer $\Pi \setminus \{\beta\}$ de $(\Sigma \setminus \Upsilon) \cup \Gamma_j \cup (\Upsilon \setminus \Theta)$ donne un ensemble de clauses satisfaisable. En effet, on a $(\Sigma \setminus \Upsilon) \cup (\Upsilon \setminus \Theta) = (\Sigma \setminus (\Theta \cup \Upsilon \setminus \Theta)) \cup (\Upsilon \setminus \Theta) = \Sigma \setminus \Theta$ et donc Π est aussi un MCS de $((\Sigma \setminus \Theta) \cup \Gamma_j)$. Cela est en contradiction avec la définition de Π qui doit être un Partial-MCS de $\langle(\Sigma \setminus \Upsilon) \cup \Gamma_j, \Upsilon \setminus \Theta\rangle$;
- $\forall \Gamma_k \in \mathcal{AC}' : (\Sigma \setminus ((\Theta \cup (\Pi \setminus \{\beta\})) \cup \{\beta\})) \cup \Gamma_k$ est satisfaisable. D'abord, remarquons que $((\Theta \cup (\Pi \setminus \{\beta\})) \cup \{\beta\}) = \Theta \cup \Sigma$. Ainsi, la quatrième hypothèse assure que $\forall \Gamma_k \in \mathcal{AC}' : (\Sigma \setminus (\Theta \cup \Pi)) \cup \Gamma_k$ est satisfaisable, ce troisième élément tient aussi.

Donc, $\forall \in \Pi$ et $\forall \Psi$ tels que Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$ et $\Psi \subseteq (\Theta \cup (\Pi \setminus \{\beta\}))) \cup \{\beta\} = \Theta \cup \Sigma$, on a ainsi $\beta \in \Psi$. Par conséquent, $\forall\ \Psi \in$ AC-MCS$(\Sigma, \mathcal{AC})$ tels que $\Psi \subseteq \Theta \cup \Pi$, alors $\Pi \subseteq \Psi$.

Exemple 5.3. Considérons à nouveau Σ et $\mathcal{AC}$ de l'exemple 5.1. Soit $\Theta = \{d \vee b\}$, $\mathcal{AC}' = \{\{\neg a\}, \{\neg e\}\}$, $\Pi = \{\neg c, \neg b\}$ et $\Gamma_j = \{\neg a\}$. Toutes les conditions de la Propriété 5.2 sont satisfaites. Ainsi, Π apparait dans chaque AC-MCS$(\Sigma, \mathcal{AC})$ inclus dans $\{d \vee b, \neg b, \neg c\}$.

Dans la prochaine section, Nous verrons comment cette propriété sera exploitée pour améliorer l'extraction d'un AC-MCS$(\Sigma, \mathcal{AC})$.

5.3 Algorithme incrémental plus élaboré

Un algorithme original pour l'extraction d'un AC-MCS$(\Sigma, \mathcal{AC})$ est illustré dans l'Algorithm 5.3, que nous allons décrire progressivement et de façon intuitive.

Par abus de notation, nous écrirons Π résout Γ_j quand la Propriété 5.2 s'applique (en référence à Π et Γ_j de la Propriété 5.2) : dans ce cas Π est un sous-ensemble de clauses de Σ qui sera inclus dans le AC-MCS final, en cours de construction.

Ψ est le AC-MCS en construction : au départ, il est initialisé à l'ensemble vide (line 1). La structure $\sigma(\Gamma_i)$ associe à chaque Γ_i un sous-ensemble de Σ. Initialement, i est affecté à 1 et ces sous-ensembles à l'ensemble vide, à l'exception de $\sigma(\Gamma_1)$, qui est initialisé à $\Sigma \setminus$ `Greedy-AC-SS`$(\Sigma, \mathcal{AC})$. Pour simplifier la compréhension, on considère pour le moment qu'il n'y a pas de pré-traitement et donc à la ligne 4, on a $\sigma(\Gamma_i) \longleftarrow \Sigma$.

Ensuite, on entre dans la boucle principale. Durant le traitement de la boucle, on fait en sorte que $\Psi \cup \bigcup_{\Gamma_i \in \mathcal{AC}} \sigma(\Gamma_i)$ soit tout le temps un AC-CS$(\Sigma, \mathcal{AC})$. Cette contrainte est bien sûr satisfaite avant d'entrer dans la boucle vu que cet ensemble est égal à Σ. Considérons la première itération de la boucle. $i = 1$ et à la ligne 7, on calcule le Partial-MCS de $\langle \Gamma_1, \Sigma \rangle$ et on stocke le résultat dans Π. Plusieurs cas se présentent. Si $\Pi = \sigma(\Gamma_i)$: dans cette première itération, nous avons alors effectivement $\Sigma = \Pi = \sigma(\Gamma_i)$. Dans ce cas de figure, toutes les conditions sont remplies pour appliquer la Propriété 5.2 et ainsi on peut insérer Π dans la solution en cours de construction Ψ (ligne 8). On affecte un ensemble vide à $\sigma(\Gamma_i)$ pour signifier que Γ_i a été résolu. i est donc décrémenté et la boucle termine étant donné que la valeur de i est égale à zéro : Ψ est le résultat renvoyé. Dans un autre cas vraiment basique, $i = m = 1$: il y a seulement un seul contexte dans $\mathcal{AC}$. De nouveau, nous avons trouvé la solution, c'est Π; mais contrairement à la situation précédente, Π doit impérativement être un sous-ensemble propre de Σ. Dans le dernier cas (*else*), on sait que Π est l'ensemble de clauses à retirer pour satisfaire Γ_i. Toutefois, certains Γ_j n'ont pas été résolus jusque là et il n'est pas certain que Π fasse entièrement partie de la solution car la Propriété 5.2 ne s'applique pas. Par conséquent, on insère $\sigma(\Gamma_i) \setminus \Pi$ dans $\sigma(\Gamma_{i+1})$ et on sauvegarde Π dans $\sigma(\Gamma_i)$. L'idée sous-jacente consiste à garder la trace dans $\sigma(\Gamma_i)$ du sous-ensemble de clauses qui aurait été suffisant pour résoudre Γ_i, tandis que les autres clauses *actuellement* restantes vont être considérées par rapport à Γ_{i+1}. Dans un certain sens, les éléments de σ sont poussés d'un cran vers l'avant. De cette manière, les clauses restantes seront considérées à la prochaine itération. Tandis qu'au même moment, on enregistre dans $\sigma(\Gamma_i)$ l'ensemble des clauses qui pourraient rendre Γ_i satisfaisable avec l'AC-MSS en construction avec $\bigcup_{k=1}^{i-1} \sigma(\Gamma_k)$.

À l'inverse, à ligne 7 quand la Propriété 5.2 s'applique, i est décrémenté. Cela conduit donc à une sorte de mouvement en arrière : Γ_i a été résolu, on peut à présent revenir en arrière et traiter Γ_{i-1} à la prochaine itération. Tout cela est réalisé de telle manière que $\sigma(\Gamma_j) = \emptyset$ pour tout $j > i$ au début et à la fin de chaque itération.

À présent, nous sommes dans une meilleure position pour comprendre le contenu de la boucle *while* et la première instruction figurant à l'intérieur (ligne 6). $\Sigma \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))$ fournit le AC-SS en cours

d'élaboration. Par construction, il est satisfaisable avec Γ_i. Maintenant, la procédure `Partial-MCS`$((\Sigma \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))) \cup \Gamma_i, \sigma(\Gamma_i))$ fournit le plus petit sous-ensemble de toutes les clauses restantes accumulées dans $\sigma(\Gamma_i)$, qui doit être supprimé de $\sigma(\Gamma_i)$ pour que ce dernier soit satisfaisable avec le premier argument de `Partial-MCS`. L'objectif est d'arriver à la situation où toutes les clauses $\sigma(\Gamma_i)$ doivent être retirées, c'est à dire, $\Pi = \sigma(\Gamma_i)$ (ligne 7), et dans ce cas, toutes les conditions sont réunies pour appliquer la Propriété 5.2. En effet, on a trouvé $\Pi =$ `Partial-MCS`$((\Sigma \setminus \Upsilon) \cup \Gamma_i, \Upsilon \setminus \Theta)$ tel que l'extension de l'AC-CS courant avec Π rendra le AC-SS correspondant satisfaisable avec tous les Γ_i, étant donné qu'il ne reste plus de clauses à tester dans $\sigma(\Gamma_i)$. Quand $i = m$, la même opération peut être appliquée : clairement, en rendant l'AC-SS courant satisfaisable avec Γ_m, on est certain que tous les Γ_i sont résolus. Dans les deux cas, on peut décrémenter i : à la prochaine itération, on tentera de résoudre Γ_{i-1} pour lequel des clauses ont été enregistrées dans $\sigma(\Gamma_{i-1})$. En effet, quand la Propriété 5.2 ne s'applique pas, on mémorise dans $\sigma(\Gamma_i)$ le sous-ensemble de clauses calculé pour résoudre Γ_i (mais qui ne permet pas de résoudre les autres Γ_j à la fois) et mettre les clauses restantes dans σ_{i+1} pour être considérées à la prochaine itération où on traitera Γ_{i+1}. À noter que quand $\sigma(\Gamma_i) = \emptyset$, l'appel à `Partial-MCS` renvoie un ensemble vide et ainsi i est décrémenté (ligne 10).

Algorithme 5.3 : `Incremental`$_2$`-AC-MCS`

Input : Σ un ensemble de clauses;
$\mathcal{AC} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_m\}$: un ensemble d'ensembles satisfaisables de clauses;
Output : Ψ t.q. Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$;

1 $\Psi \longleftarrow \emptyset$;
2 σ associe pour chaque $\Gamma_i \in \mathcal{AC}$ un ensemble de clauses, initialement tous vides;
3 $i \longleftarrow 1$;
4 $\sigma(\Gamma_i) \longleftarrow \Sigma \setminus$ `Greedy-AC-SS`$(\Sigma, \mathcal{AC})$;
5 **while** $i > 0$ **do**
6 $\quad \Pi \longleftarrow$ `Partial-MCS`$((\Sigma \setminus (\Psi \cup \bigcup_{j=1}^{i} \sigma(\Gamma_j))) \cup \Gamma_i,\ \sigma(\Gamma_i))$;
7 $\quad$ **if** $\Pi = \sigma(\Gamma_i)$ **or** $i = m$ **then**
8 $\quad\quad \Psi \longleftarrow \Psi \cup \Pi$;
9 $\quad\quad \sigma(\Gamma_i) \longleftarrow \emptyset$;
10 $\quad\quad i \longleftarrow i - 1$;
11 $\quad$ **else**
12 $\quad\quad \sigma(\Gamma_{i+1}) \longleftarrow \sigma(\Gamma_i) \setminus \Pi$;
13 $\quad\quad \sigma(\Gamma_i) \longleftarrow \Pi$;
14 $\quad\quad i \longleftarrow i + 1$;
15 **return** Ψ;

Exemple 5.4. Déroulons `Incremental`$_2$`-AC-MCS` sur l'exemple 5.1, où $\Gamma_1 = \{\neg a\}$, $\Gamma_2 = \{\neg e\}$ et $\Gamma_3 = \{\neg d\}$. Soit $\{a \vee b, a \vee c, e \vee c\}$, le AC-SS renvoyé par `Greedy-AC-SS`. Avant d'entrer dans la boucle, nous avons $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c, d \vee b\}$ et $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

- Itération $\#1$: $\Pi = \{\neg b, \neg c\}$. Puisque $\Pi \neq \sigma(\Gamma_1)$ et $i \neq 3$, on rentre dans la partie `else`. Ainsi, $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \{d \vee b\}$ et $\sigma(\Gamma_3) = \emptyset$.
- Itération $\#2$: $\Pi = \emptyset$. Idem à la première itération, la partie `else` est considérée. Ainsi, $i = 3$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$, $\sigma(\Gamma_2) = \emptyset$ et $\sigma(\Gamma_3) = \{d \vee b\}$.
- Itération $\#3$: $\Pi = \emptyset$. Comme $i = n$, la partie `if` est exécutée. Ainsi, $i = 2$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ et $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

- Itération $\#4$: $\Pi = \emptyset$. Comme $\sigma(\Gamma_2) = \emptyset$, la partie `if` est considérée. Ainsi, $i = 1$, $\Psi = \emptyset$, $\sigma(\Gamma_1) = \{\neg b, \neg c\}$ et $\sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.
- Itération $\#5$: $\Pi = \{\neg b, \neg c\}$, $\Pi = \sigma(\Gamma_1)$. La partie `if` est exécutée. Ainsi, $i = 0$, $\Psi = \{\neg b, \neg c\}$, $\sigma(\Gamma_1) = \sigma(\Gamma_2) = \sigma(\Gamma_3) = \emptyset$.

L'algorithme sort de la boucle et retourne Ψ, qui est un AC-MCS$(\Sigma, \mathcal{AC})$.

Propriété 5.3. `Incremental`$_2$`-AC-MCS`$(\Sigma, \mathcal{AC})$ renvoie toujours un AC-MCS$(\Sigma, \mathcal{AC})$. Dans le pire cas, cette procédure nécessite $\mathcal{O}(nm)$ appels à `Partial-MCS`, où $n = \mathit{card}(\Sigma)$ et $m = \mathit{card}(\mathcal{AC})$.

Preuve 5.3. D'abord, introduisons quelques notations pratiques pour décrire les valeurs de variables principales au début de la $x^{\mathit{ième}}$ boucle (lignes 6–15). On définit i_x (respectivement, Ψ_x et σ_x) comme valeur de i (respectivement, Ψ et σ) au début de la $x^{\mathit{ième}}$ boucle. Initialement, $i_0 = 1$, $\Psi_0 = \emptyset$, $\sigma_0(\Gamma_1) = \Sigma$ et $\forall j \in [2, m]$ (où $m = \mathit{card}(\mathcal{AC})$) : $\sigma_0(\Gamma_j) = \emptyset$. On note aussi par Π_x le Partial-MCS calculé à la $x^{\mathit{ième}}$ itération (ligne 7). Maintenant, montrons par induction sur le nombre d'itérations que $\forall j > i$: $\sigma(\Gamma_j) = \emptyset$ (1) est un invariant de la boucle. Comme indiqué précédemment, nous avons $\forall j \in [2, m]$: $\sigma_0(\Gamma_j) = \emptyset$. Ainsi, l'affirmation est vraie pour $x = 0$. Montrons que, en supposant que l'affirmation (1) soit vraie pour x, elle est aussi vraie pour $x+1$. Considérons les deux situations différentes où les valeurs de σ et i ont changé. D'abord, considérons le cas où $\sigma_x(\Gamma_i)$ reçoit $\emptyset$ et i_x est décrémenté (lignes 10–11), il est facile de voir que l'énoncé est toujours valable. Considérons maintenant le seconde situation (lignes 13–15) : nous constatons que $i_{x+1} = i_x + 1$, et $\sigma_{i_{x+1}}(\Gamma_{i_{x+1}-1}) = \sigma_{i_x}(\Gamma_{i_x})$ et $\sigma_{i_{x+1}}(\Gamma_{i_{x+1}}) = \sigma_{i_x}(\Gamma_{i_x+1})$ sont les seules valeurs qui ont changé. Ainsi, nous avons $\forall j > i_x + 1$, $\sigma_{i_{x+1}}(\Gamma_j) = \sigma_{i_x}(\Gamma_j) = \emptyset$ par hypothèse d'induction et donc $\forall j > i_{x+1} : \sigma_{i_{x+1}}(\Gamma_j) = \emptyset$. Par conséquent, l'affirmation (1) est vraie.

Maintenant, montrons qu'au début de chaque boucle d'itération nous avons $\Psi \cup \bigcup_{k=1}^{m} \sigma(\Gamma_k)$ qui est AC-CS$(\Sigma, \mathcal{AC})$ (2). Par induction sur le nombre d'itérations x. Pour $x = 0$, nous avons $\Sigma \setminus (\Psi_0 \cup \bigcup_{k=1}^{m} \sigma_0(\Gamma_k)) = \Sigma \setminus (\emptyset \cup \Sigma) = \emptyset$, qui est satisfaisable avec chaque contexte de $\mathcal{AC}$. Par conséquent, l'affirmation (2) est vraie pour $x = 0$. Considérons l'étape $x + 1$ et supposons que (2) est vraie $\forall x' \in \mathbb{N}$ tel que $x' \leq x$. Nous devons examiner les deux cas où σ est modifié.

Dans le premier cas (lignes 13 – 14), nous avons $i_{x+1} = i_x + 1$, $\sigma_{x+1}(\Gamma_{i_{x+1}}) = \sigma_x(\Gamma_{i_x}) \setminus \Pi$, $\sigma_{x+1}(\Gamma_{i_{x+1}-1}) = \Pi$ et $\forall j \in \{1, \ldots, i_{x+1} - 2, i_{x+1} + 1, \ldots, m\} : \sigma_{x+1}(\Gamma_j) = \sigma_x(\Gamma_j)$. Ainsi, nous avons

$$
\begin{aligned}
\bigcup_{k=1}^{m} \sigma_{x+1}(\Gamma_k) &= \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) && \text{de (1)} \\
&= \bigcup_{k=1}^{i_{x+1}-2} \sigma_{x+1}(\Gamma_k) \cup \sigma_{x+1}(\Gamma_{i_{x+1}-1}) \cup \sigma_{x+1}(\Gamma_{i_{x+1}}) \\
&= \bigcup_{k=1}^{i_{x+1}-2} \sigma_x(\Gamma_k) \cup \sigma_{x+1}(\Gamma_{i_{x+1}-1}) \cup \sigma_{x+1}(\Gamma_{i_{x+1}}) \\
&= \bigcup_{k=1}^{i_{x+1}-2} \sigma_x(\Gamma_k) \cup \Pi \cup (\sigma_x(\Gamma_{i_x}) \setminus \Pi) \\
&= \bigcup_{k=1}^{i_x-1} \sigma_x(\Gamma_k) \cup \sigma_x(\Gamma_{i_x}) = \bigcup_{k=1}^{i_x} \sigma_x(\Gamma_k) \\
&= \bigcup_{k=1}^{m} \sigma_x(\Gamma_k) && \text{de (1)}
\end{aligned}
$$

Donc, (2) est valide dans cette première situation.

Considérons le cas où $\Pi = \sigma_x(\Gamma_{i_x})$ ou $i_x = m$. Si $\Pi = \sigma_x(\Gamma_{i_x})$ alors $i_{x+1} = i_x$-1 et σ_{x+1} est différent de σ_x uniquement sur la valeur de $\sigma_{x+1}(\Gamma_{i_x})$, qui est affecté à $\emptyset$. Ainsi, nous avons

$$\begin{aligned}\Psi_x \cup \bigcup_{k=1}^{i_x} \sigma_x(\Gamma_k) &= \Psi_x \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \cup \sigma_x(\Gamma_{i_x}) \\ &= \Psi_x \cup \sigma_x(\Gamma_{i_x}) \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \\ &= \Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k)\end{aligned}$$

qui est donc, par induction, un AC-CS (Σ,$\mathcal{AC}$). Maintenant examinons le cas où $i_s = m$. Supposons que $\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \cup \Pi$ n'est pas un AC-CS$(\Sigma, \mathcal{AC})$: $\exists \Gamma_\ell \in \mathcal{AC}$ tel que $\Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \cup \Pi) \cup \Gamma_\ell$ est insatisfaisable. Par la définition d'un Partial-MCS, nous avons nécessairement $\ell < m$. Considérons x' la dernière fois où $i'_x = \ell$. Comme la valeur de i peut uniquement être incrémentée et décrémentée, il en résulte que pour deux itérations (arbitraires) t_1 et t_2 de boucle toutes les valeurs entre i_{t_1} et i_{t_2} doivent être considérées au moins une fois. Ainsi, il est facile de voir que pour tout tour de boucle $x' \leq t_1 < t_2 \leq x+1$ nous avons $\sigma_{t_1}(\Gamma_j) = \sigma_{t_2}(\Gamma_j)$ pour tout $j \leq \ell$ (autrement, $\exists x'' > x'$ tel que $i_{x''} < i_{x'}$ et dont x' n'est pas la dernière fois où $i = \ell$). De plus, à l'étape x' la condition `if` n'est pas satisfaite (autrement $i_{x'+1} < i_{x'}$). Ainsi, $\Psi_{x'+1} = \Psi_{x'}$ et $\sigma_{x'+1}$ est calculé de telle sorte que $\Sigma \setminus (\Psi_{x'} \cup \bigcup_{k=1}^{\ell} \sigma_{x'}(\Gamma_k) \cup \sigma_{x'+1}(\Gamma_\ell)) \cup \Gamma_\ell$ soit satisfaisable. Par construction, $\bigcup_{k=1}^{\ell} \sigma_{x'+1}(\Gamma_k) = \bigcup_{k=1}^{\ell-1} \sigma_{x'}(\Gamma_k) \cup \sigma_{x'+1}(\Gamma_\ell)$ (ligne 13) et Ψ peut uniquement être augmenté à la ligne 9. Donc, nous avons $\Psi_{x'+1} \cup \bigcup_{k=1}^{\ell} \sigma_{x'+1}(\Gamma_k) \subseteq \Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \cup \Pi$ et donc $\Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k) \cup \Pi) \cup \Gamma_\ell$ est satisfaisable. Cela contredit l'hypothèse précédente, et donc (2) est vérifié.

Maintenant, démontrons un autre invariant de la boucle : $\Gamma_j \cup \Sigma \setminus (\Psi \cup \bigcup_{k=1}^{j} \sigma(\Gamma_k))$ est satisfaisable $\forall j \leq i$ (3). Par induction sur le nombre d'itérations de la boucle. Comme $i_0 = 1$, $\Psi_0 = \emptyset$ et $\sigma_0(\Gamma_1) = \Sigma$, l'affirmation (3) est vraie pour $s = 0$. Étant donné $x \in \mathbb{N}$ et supposons que (3) est vraie pour x. Montrons qu'il est encore pour $x+1$, c'est-à-dire, que $\Gamma_j \cup \Sigma \setminus (\Psi_{s+1} \cup \bigcup_{k=1}^{j} \sigma_{s+1}(\Gamma_k))$ est satisfaisable $\forall j \leq i_{s+1}$. Considérons les deux cas déclenchés par le `if`. D'abord, lorsque la condition du `if` est satisfaite (lignes 8–11), la valeur de $i_{x+1} = i_x - 1$ et la valeur de $\sigma_{x+1}(\Gamma_k) = \sigma_x(\Gamma_k)$ pour tout $k < i_x - 1$. Ainsi, par induction, on déduit que le Γ_{x+1} satisfait (3). Lorsque la condition du `if` est falsifiée, nous avons $i_{x+1} = i_x + 1$ et uniquement les valeurs de $\sigma_{x+1}(\Gamma_{i_x})$ et $\sigma_{x+1}(\Gamma_{i_x+1})$ qui diffèrent de σ_x. Donc pour montrer que l'énoncé est encore vrai, il suffit de montrer que $\Gamma_{i_{x+1}-1} \cup \Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}-1} \sigma_{x+1}(\Gamma_k))$ sont satisfaisables. Puisque $\Gamma_{i_{x+1}-1} \cup \Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}-1} \sigma_{x+1}(\Gamma_k)) = \Gamma_{i_x} \cup \Sigma \setminus (\Psi_x \cup \bigcup_{k=1}^{i_x-1} \sigma_x(\Gamma_k) \cup \sigma_{x+1}(\Gamma_{i_x}))$, donc comme $\sigma_{s+1}(\Gamma_{i_s})$ est un ensemble rectificatif de $\Gamma_{i_s} \cup \Sigma \setminus (\Psi_s \cup \bigcup_{k=1}^{i_s} \sigma_s(\Gamma_k))$, nous avons $\Gamma_{i_{x+1}-1} \cup \Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}-1} \sigma_{x+1}(\Gamma_k))$ qui est satisfait. Maintenant, montrons que $\Gamma_{i_{x+1}} \cup \Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k))$ est satisfaisable. Comme $\Psi_{x+1} \cup \bigcup_{k=1}^{i_x+1} \sigma_{x+1}(\Gamma_k) = \Psi_x \cup \bigcup_{k=1}^{i_x} \sigma_x(\Gamma_k) = \Psi_x \cup \bigcup_{k=1}^{m} \sigma_x(\Gamma_k)$, donc d'après (2) nous avons $\Psi_{x+1} \cup \bigcup_{k=1}^{i_x+1} \sigma_{x+1}(\Gamma_k)$ qui est un AC-CS(Σ, Γ) et donc $\Gamma_{i_{x+1}} \cup \Sigma \setminus (\Psi_{x+1} \cup \bigcup_{k=1}^{i_{x+1}} \sigma_{x+1}(\Gamma_k))$ est satisfaisable. Donc, par induction (3) est vrai.

Démontrons, encore une fois par induction, qu'à chaque étape x, $\forall\ \Psi'$ tel que Ψ' est un AC-MCS $(\Sigma, \mathcal{AC})$ tel que $\Psi' \subseteq \Psi_x \cup \bigcup_{k=1}^{m} \sigma_x(\Gamma_k)$, nous avons $\Psi_x \subseteq \Psi'$ (4). À l'étape initiale $\Psi_0 = \emptyset$ est donc l'affirmation (4) est vraie pour $s = 0$. Maintenant, montrons qu'à l'étape $x+1$ nous avons l'affirmation (4) qui est vraie, en supposant qu'elle est vraie à l'étape x. Pour ce faire, nous avons besoin de traiter

trois situations différentes. La première : $\Pi_s = \sigma_s(\Gamma_{i_s})$. Il est facile de voir que l'affectation de Υ à $\Psi_x \cup \bigcup_{k=1}^{i_x} \sigma_x(\Gamma_k)$, Π à Π_x, $\mathcal{AC}'$ to $\{\Gamma_{i_x}, \ldots, \Gamma_m\}$ et Γ à Γ_{i_x} permet de satisfaire toutes les conditions de la Propriété 5.2 pour qu'elle soit appliquée. Donc, si $\Psi' \subseteq \Psi_x \cup \bigcup_{k=1}^{m} \sigma_x(\Gamma_k) = \Psi_x \cup \Pi_x \cup \bigcup_{k=1}^{m} \sigma_{x+1}(\Gamma_k)$ alors nous avons $\Pi_x \subseteq \Psi'$. Par l'hypothèse d'induction, nous avons déjà $\Psi_x \subseteq \Psi'$. Ainsi, $\Psi_x \cup \Pi_x = \Psi_{x+1} \subseteq \Psi'$ et l'affirmation (4) est vraie dans le premier cas. Maintenant, considérons le deuxième cas où $i_x = m$. Donc, si nous affectons Υ à $\Psi_s \cup \bigcup_{k=1}^{m} \sigma_s(\Gamma_k)$, Φ à $\Psi_x \cup \bigcup_{k=1}^{m-1} \sigma_x(\Gamma_k)$, Π à Π_x, $\mathcal{AC}'$ à $\{\Gamma_n\}$ et Γ à Γ_m, toutes les conditions de la Propriété 5.2 sont vérifiées. Donc, $\forall\ \Psi'$ tel que Ψ' est un AC-MCS(Σ, Γ) tel que $\Psi' \subseteq \Psi_x \cup \Pi_x \cup \bigcup_{k=1}^{m-1} \sigma_x(\Gamma_k) = \Psi_x \cup \Pi_x \cup \bigcup_{k=1}^{m-1} \sigma_{x+1}(\Gamma_k)$. De plus, par induction nous avons $\Psi_x \subseteq \Psi'$. Donc, $\Psi_x \cup \Pi_x = \Psi_{x+1} \subseteq \Psi'$ et l'affirmation (4) est vraie dans le deuxième cas. Enfin, considérons le troisième et dernier cas (lignes 13–15). Puisque $\Psi_{x+1} = \Psi_x$ et $\bigcup_{k=1}^{m} \sigma_x(\Gamma_k) = \bigcup_{k=1}^{m} \sigma_{x+1}(\Gamma_k)$, par induction, l'affirmation (4) est vraie.

À présent, montrons que si $i \leq 0$ alors Ψ est un AC-MCS$(\Sigma, \mathcal{AC})$. De (1), nous avons $i \leq 0$, donc $\bigcup_{k=1}^{m} \sigma(\Gamma_k) = \emptyset$. En outre, de (4) nous savons que $\forall \Psi'$ tel que Ψ' est AC-MCS$(\Sigma, \mathcal{AC})$ tel que $\Psi' \subseteq \Psi \cup \bigcup_{k=1}^{m} \sigma(\Gamma_k)$, nous avons $\Psi \subseteq \Psi'$. De ce fait, $\Psi' \subseteq \Psi \cup \bigcup_{k=1}^{m} \sigma(\Gamma_k) = \Psi$ et $\Psi \subseteq \Psi'$, et donc $\Psi \in$ AC-MCS$(\Sigma, \mathcal{AC})$.

Enfin, montrons que, après un nombre fini d'itérations, l'algorithme se termine et que l'algorithme nécessite, dans le pire cas, un nombre d'appels à `Partial-MCS` quadratique en *card*$(\mathcal{AC})$. La boucle termine quand $i \leq 0$. Notons que lorsque $\bigcup_{k=1}^{m} \sigma(\Gamma_k) = \emptyset$, Π est égal à $\emptyset$ et i est décrémenté, tandis que $\bigcup_{k=1}^{m} \sigma(\Gamma_k) = \emptyset$ ne change pas. Cela suffit pour montrer que la boucle se termine toujours dans le cas spécifique où $\bigcup_{k=1}^{m} \sigma(\Gamma_k) = \emptyset$. Dans les autres cas, comme $\forall\ \sigma(\Gamma_i)$ nous avons $\sigma(\Gamma_i)$ qui est augmenté uniquement avec les clauses de $\bigcup_{k=1}^{m} \sigma(\Gamma_k)$, et $\bigcup_{k=1}^{m} \sigma(\Gamma_k)$ ne peut pas être étendu entre deux itérations. Ensuite, il suffit de démontrer cela à partir de n'importe quelle étape de boucle t_1 telle que $\bigcup_{k=1}^{m} \sigma_{t_1}(\Gamma_k) \neq \emptyset$, il y aura une étape t_2 avec $t_1 < t_2$, où $\bigcup_{k=1}^{m} \sigma_{t_2}(\Gamma_k) \subset \bigcup_{k=1}^{m} \sigma_{t_1}(\Gamma_k)$. Du fait que $\bigcup_{k=1}^{m} \sigma_{t_1}(\Gamma_k) \neq \emptyset$, nous pouvons admettre que, sans perdre de généralité, que $\sigma_{t_1}(\Gamma_{i_{t_1}}) \neq \emptyset$. Considérons l'étape t'_1 initialement égale à t_1, si $\Pi_{t'_1} = \sigma_s(\Gamma_{i_{t'_1}})$ ou $i_{t'_1} = m$, alors à $t_2 = t'_1 + 1$ $\bigcup_{k=1}^{m} \sigma_{t_2}(\Gamma_k) \subset \bigcup_{k=1}^{m} \sigma_{t'_1}(\Gamma_k)$. Sinon, puisque $\sigma_{t'_1}(\Gamma_{i_{t'_1}}) \neq \Pi_s$, nous avons $\sigma_{t'_1+1}(\Gamma_{i_{t'_1+1}}) \neq \emptyset$. Ainsi, nous pouvons réaliser ce processus avec $t'_1 = t'_1 + 1$ jusqu'à ce que $\Pi_{t'_1} = \sigma_s(\Gamma_{i_{t'_1}})$, ou $i_{t'_1} = m$ (en fait, nous avons seulement besoin de $m - i_{t_1}$ étapes pour atteindre un état où au moins une clause de l'union des $\sigma(\Gamma_j)$ est supprimée). Par conséquent, dans le pire des cas, après un certain nombre d'itérations qui est quadratique en *card*$(\mathcal{AC})$, l'algorithme termine.

Notons qu'initialiser $\sigma(\Gamma_1)$ à un AC-CS $(\Sigma, \mathcal{AC})$ à la ligne 4 ne change rien à la correction de l'algorithme. En effet, par construction, nous avons $\Psi_0 \cup \bigcup_{k=1}^{m} \sigma_0(\Gamma_k)$ qui est un AC-CS$(\Sigma, \mathcal{AC})$ et $\Gamma_j \cup \Sigma \setminus (\Psi \cup \bigcup_{k=1}^{j} \sigma_0(\Gamma_k))$ est satisfaisable $\forall j \leq i_0$. Ainsi, tous les invariants de boucle établis dans la preuve sont toujours valides avant d'entrer dans la boucle principale lorsque $\sigma(\Gamma_1) \leftarrow$ AC-CS$(\Sigma, \mathcal{AC})$ (ligne 4). Par conséquent, la preuve ci-dessus reste vraie même lorsque $\sigma(\Gamma_1)$ est initialisé à un AC-CS $(\Sigma, \mathcal{AC})$.

Soulignons que la structure de la procédure `Incremental`$_2$`-AC-MCS` permet de profiter des spécificités avantageuses d'un solveur SAT incremental. À cet égard, nous avons implanté la procédure `Partial-MCS` en utilisant une technique similaire à celle qui est présentée dans (Audemard *et al.* 2013) pour GLUCOSE (Audemard et Simon 2014) afin d'obtenir un système qui réutilise autant que possible les recherches effectuées par le solveur SAT pendant les appels précédents (à l'intérieur d'un même appel à `Partial-MCS` et d'un appel à l'autre). Soulignons aussi que cette implantation est caractérisée par d'autres optimisations. En particulier, dans la boucle principale, les appels inutiles à `Partial-MCS`$(\ldots, \emptyset)$ sont évités. De plus, notre implantation bénéficie des récentes avancées dans le calcul MSS et MCS, telles que celles proposées dans (Grégoire *et al.* 2014a) et utilisées dans (Besnard *et al.* 2015) pour l'algorithme `Transformational-AC-MSS`.

5.4 Expérimentations

Nous avons confronté les trois approches : `Incremental`$_1$`-AC-MSS`, `Incremental`$_2$`-AC-MCS` et `Transformational-AC-MSS` de manière pratique. Pour ce faire, nous avons sélectionné un ensemble de 295 benchmarks insatisfaisables de la dernière compétition d'extraction de MUS [9]. Ces benchmarks représentent les instances de Σ. Nous avons choisi différentes valeurs pour $card(\mathcal{AC})$: 2, 5, 10, 15, 20, 25, 30, 35, 40, 45 et 50 : ce paramètre représentant les différents nombres de contextes. Puis, en utilisant les variables de Σ, nous avons généré aléatoirement chaque Γ_i, de sorte à avoir un ensemble satisfaisable et formé de 50 clauses binaires. Nous avons fait exécuter nos expérimentations sur des processeurs Intel Xeon E5-2643 (3.30GHz) avec 8G octets de mémoire vive et sous Linux CentOS. Nous avons fixé le *timeout* à 900 secondes par instance et par test. Nous avons exécuté l'outil de `Transformational-AC-MSS` disponible à l'adresse `http://www.cril.fr/AAAI15-BGL`. Nous avons implanté tous les autres algorithmes en C++ sous GLUCOSE [10]. Notons que notre outil ainsi que toutes les données et tous les résultats de ces expérimentations sont disponibles à l'adresse suivante : `http://www.cril.fr/AAAI16-GIL`. La constante *iteration-max* dans `Greedy-AC-SS` fut fixée à 10.

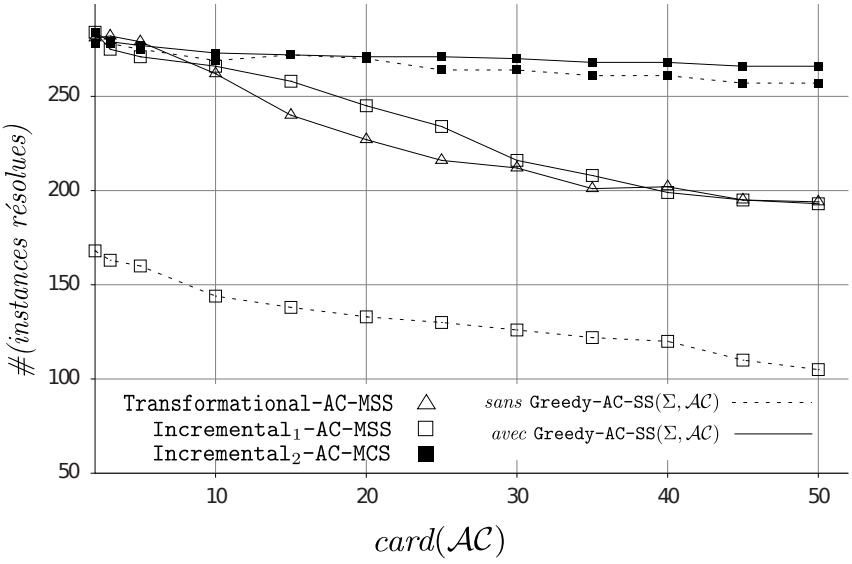

FIGURE 5.1 – Nombre d'instances résolues.

Les résultats montrent clairement qu'`Incremental`$_2$`-AC-MCS` est meilleur que les autres approches. Comme l'illustre la figure 5.1, pour chaque valeur de $card(\mathcal{AC})$, `Incremental`$_2$`-AC-MCS` est celle qui résout le plus d'instances par rapport aux autres méthodes testées ; la différence croît en fonction de $card(\mathcal{AC})$. De plus, les résultats ne diffèrent pas lorsque `Incremental`$_2$`-AC-MCS` n'inclut pas `Greedy-AC-SS`, ce qui montre que les bonnes performances de l'algorithme le sont grâce aux propriétés de AC-MCS développées dans cette étude. `Incremental`$_1$`-AC-MSS` se révèle moins bon, mais avec l'appel à `Greedy-AC-SS` il arrive à dépasser `Transformational-AC-MSS` en termes d'instances résolues. Le nombre d'instances résolues par `Incremental`$_2$`-AC-MCS`, compris entre 266 et 284 (sur un total de 295 instances testées), dépendent de $card(\mathcal{AC})$. De même, la plupart des instances sont résolues de manière plus rapide avec `Incremental`$_2$`-AC-MCS`. La figure 5.2 compare cette dernière avec l'`approche de Transformation` en terme de temps CPU en secondes pour résoudre

9. `http://www.cril.univ-artois.fr/SAT11/results/results.php?idev=48`
10. `http://www.labri.fr/perso/lsimon/glucose/`

chaque instance, pour chaque valeur de $card(\mathcal{AC})$. La figure 5.3 affiche des résultats similaires quand Incremental$_2$-AC-MCS et Incremental$_1$-AC-MSS sont comparés.

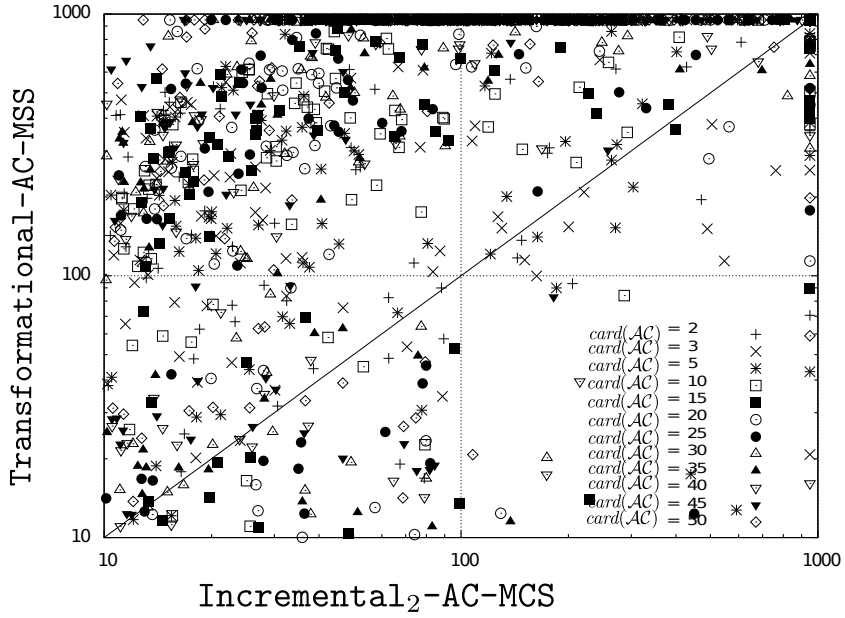

FIGURE 5.2 – Comparaison des temps de résolution (en secondes) entre Incremental$_2$-AC-MCS et Approche de Transformation.

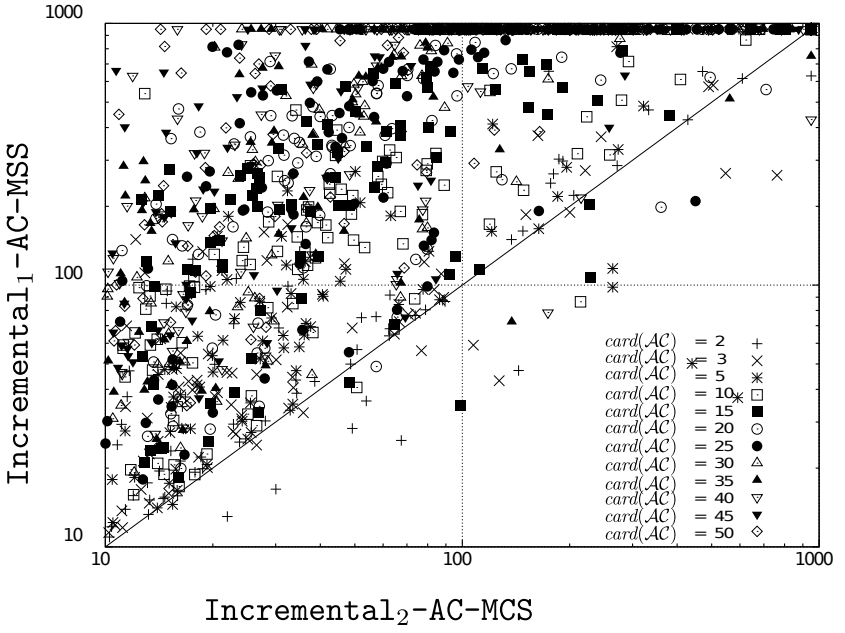

FIGURE 5.3 – Comparaison des temps de résolution (en secondes) entre Incremental$_2$-AC-MCS et Incremental$_1$-AC-MSS.

5.5 Conclusion

La recherche d'un sous-ensemble d'informations satisfaisable avec une série de contextes, éventuellement mutuellement contradictoires, est un point central dans beaucoup de domaines de l'Intelligence Artificielle. Dans cette étude, nous avons proposé une méthode originale d'extraction d'un tel sous-ensemble en logique propositionnelle. Nos expérimentations montrent que cette méthode obtient les meilleurs résultats, comparée aux approches déjà proposées.

Les résultats obtenus ouvrent la voie à de nombreuses perspectives. Clairement, la procédure d'extraction d'un Partial-MCS qui exploite les techniques des solveurs SAT incrémentaux peut aussi être améliorée avec des optimisations spécifiques utilisées par les méthodes d'extraction de MCS, comme celle qui est proposée dans (Mencía *et al.* 2015). Également, notre méthode peut constituer la brique élémentaire d'une méthode d'énumération des sous-ensembles d'information maximaux non contradictoire avec une série de contextes possibles (du moins quand cette tâche est réalisable en pratique). La principale difficulté qui réside dans une telle méthode est de savoir comment réutiliser les informations obtenues dans la recherche d'un sous-ensemble, dans l'extraction du prochain sous-ensemble. Enfin, orienter ce travail vers une méthode *incrémentale directe* d'extraction d'un sous-ensemble maximal par rapport à la cardinalité qui soit cohérent avec de multiples contextes demeure un véritable défi.

Chapitre

6

Calcul d'un consensus $max_{\subseteq}$

Sommaire

La recherche d'un consensus qui réconcilie plusieurs agents conflictuels ou sources d'informations est un paradigme clé dans beaucoup de domaines de l'intelligence artificielle. Nous avons vu dans le chapitre 3 qu'il existe de nombreuses définitions du concept de consensus (voir, par exemple, (Ephrati et Rosenschein 1996, Ren *et al.* 2005, Jøsang 2002, Gauwin *et al.* 2007)). Dans cette étude, nous complétons quelques travaux récents (Grégoire *et al.* 2016c) où ce concept a été défini et étudié dans le cadre de la logique propositionnelle. L'idée principale est qu'un consensus réconcilie plusieurs agents ou sources d'informations en conflit et devrait être un sous-ensemble non contradictoire de toutes les informations à concilier. Pour former un consensus, ce sous-ensemble doit être non-conflictuel avec chacun des agents ou sources, pris individuellement. Ce concept de consensus traduit donc une attitude libérale des agents impliqués, dans le sens où, un agent est prêt à accepter la présence dans le consensus de certaines informations qui ne sont pas issues de ses propres prémisses. D'autre part, il est assurée que ses propres informations ne seront contredites logiquement par aucun consensus.

Dans (Grégoire *et al.* 2016c), divers critères de maximalité dans les consensus ont été étudiés. En particulier, une méthode a été proposée pour calculer un consensus présentant la cardinalité maximale possible, à savoir un consensus $max_{\#}$. Comme tout consensus $max_{\#}$ est aussi un $max_{\subseteq}$, cette méthode délivre ainsi un consensus $max_{\subseteq}$.

Soulignons ici que toute méthode permettant de calculer un consensus doit résoudre les principaux problèmes de calcul suivants. Tout consensus doit être satisfaisable avec chaque source. Comme les sources peuvent être mutuellement conflictuelles, la conjonction de toutes les sources peut alors être insatisfaisbale. Donc, vérifier si un consensus candidat est satisfaisable avec chaque source séparément ne peut pas se ramener à tester directement si ce consensus est satisfaisable avec la conjonction des sources. Ce problème est contourné dans Grégoire *et al.* (2016c) en utilisant un schéma de transformation qui réécrit le problème de recherche de consensus de telle sorte que le test de satisfaisabilité puisse être effectué avec toutes les sources (réécrites) ensemble. De plus, l'approche par transformation implique un calcul en une seule étape qui garantit que le consensus soit maximal par rapport à la cardinalité. L'approche par transformation a été démontrée expérimentalement plus efficace (en temps de calcul) qu'une méthode incrémentale directe basée sur des itérations de tests de satisfaction pour : (i) vérifier la consistance avec chaque source et (ii) assurer la maximalité par rapport à la cardinalité (Grégoire *et al.* 2016c).

Les consensus $max_{\subseteq}$ ont clairement un intérêt spécifique car ils sont des sous-ensembles de l'ensemble des informations des agents qui ne contredisent aucun agent et qui sont maximaux, dans le sens où l'ajout de toute formule supplémentaire de n'importe quel agent contredirait au moins un agent. Rappelons que bien qu'il puisse y avoir un nombre exponentiel de consensus $max_{\#}$ et $max_{\subseteq}$ dans le pire des

cas, *un* consensus $max_{\subseteq}$ est souvent suffisant dans de nombreux problèmes d'application impliquant la négociation entre agents. Cela peut être également un point de départ utile pour les négociations futures.

Dans ces travaux, nous nous intéressons au calcul d'un consensus $max_{\subseteq}$ et la question posée est la suivante : peut-on concevoir une méthode incrémentale qui serait significativement plus efficace en termes de temps et d'espace que la méthode par transformation pour calculer un consensus $max_{\subseteq}$? Nous répondons à cette question de manière positive : nous développons et expérimentons un algorithme spécifique qui s'avère expérimentalement beaucoup plus efficace que l'approche par transformation proposée dans l'étude initiale, même lorsque l'appel au solveur Partial-MAX-SAT est remplacé par un appel au solveur Partial-MCS dans la méthode par transformation. Ces travaux ont donné lieu à une publication internationale (Grégoire *et al.* 2017).

Dans ce chapitre, nous mettons en lumière le lien qui existe entre les deux concepts consensus et AC-MSS. Puis, nous présentons la méthode par transformation pour le calcul d'un consensus $max_{\subseteq}$ et la méthode incrémentale la plus performante pour la recherche d'un AC-MSS. La comparaison expérimentale entre les deux méthodes dans le contexte du calcul d'un consensus $max_{\subseteq}$ est ensuite effectuée sur un large panel de benchmarks, avant de finir par une conclusion.

6.1 Consensus et AC-MSSes

Le concept d'AC-MSS a été défini dans (Besnard *et al.* 2015) afin de capturer une forme de planification minutieuse sous un éventail d'hypothèses potentiellement contradictoires qu'un agent émet sur ce qu'il ne connait pas. Dans ce contexte, un agent sceptique pourrait vouloir considérer (et appuyer ses décisions) uniquement ce qui ne peut être contredit si certaines de ses hypothèses s'avèrent finalement être vraies. Ainsi, lorsque Γ est un AC-MSS$(\Theta, \mathcal{AC})$ (avec Θ une CNF satisfaisable représentant la base de connaissance d'un agent et $\mathcal{AC}$ un ensemble contextes hypothétique), il représente un sous-ensemble des connaissances et croyances Θ d'un agent. Ce sous-ensemble n'est pas en conflit avec des contextes hypothétiques que l'agent envisage comme possibles. Il est maximal dans le sens où aucun sur-ensemble strict de Γ dans Θ affiche la même propriété. Ainsi, l'agent peut fonder ses décisions sur Γ, il ne sera pas contredit si un contexte hypothétique s'avère être vrai, en particulier dans le futur. Curieusement, le problème de calcul de consensus $max_{\subseteq}$, techniquement parlant, coïncide avec l'extraction d'un AC-MSS. En effet, bien que les deux concepts de consensus (consensus $max_{\subseteq}$ et AC-MSS) ont été définis pour des applications différentes, il est facile de remarquer qu'il y a une correspondance entre les deux définitions.

Propriété 6.1. Γ est un consensus $max_{\subseteq}$ du profile $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$ si et seulement si Γ est un AC-MSS$(\bigcup_{i=1}^{n} \Phi_i, \mathcal{S})$.

Les techniques développées pour le calcul d'un AC-MSS peuvent ainsi être utilisées pour l'extraction d'un consensus $max_{\subseteq}$. Ces techniques peuvent ensuite être comparées expérimentalement avec la méthode par transformation utilisée par (Grégoire *et al.* 2016c) dans le contexte de la recherche d'un consensus $max_{\subseteq}$.

6.2 Calcul d'un consensus $max_{\subseteq}$

Dans cette section, nous instancions la méthode générique appelée "méthode par transformation" proposée dans (Grégoire *et al.* 2016c) pour cibler la maximalité par rapport à l'inclusion ($max_{\subseteq}$). Nous allons ensuite décrire brièvement comment appliquer la méthode de calcul d'un AC-MSS proposé dans (Grégoire *et al.* 2016b), pour fournir un consensus $max_{\subseteq}$ avec un ensemble de contraintes d'intégrité que le consensus doit satisfaire (inclure).

Méthode par transformation

La méthode par transformation (Grégoire *et al.* 2016c) pour le calcul d'un consensus $max_{\#}$ consiste à faire un appel à un solveur Partial-MAX-SAT sur une instance transformée du problème. Pour calculer un consensus $max_{\subseteq}$ (maximalité par rapport à l'inclusion ensembliste et non par rapport à la cardinalité), on utilise un solveur Partial-MSS à la place d'un solveur Partial-MAX-SAT.

L'instanciation de l'approche générique reporté dans l'algorithme 3.1 (voir section 3.2) donne lieu à l'algorithme 6.1 qui prend aussi en compte les contraintes d'intégrité Δ. Le processus reporté dans l'algorithme 6.1, consiste à transformer (réécrire) en un sous-problème indépendant chaque condition de satisfaisabilité de $\Gamma \cup \Phi_i \cup \Delta$, en utilisant des variables fraîches. Les sous-problèmes sont regroupés pour former un seul problème global en utilisant des variables partagées et d'autres nouvelles variables. Le calcul du consensus $max_{\subseteq}$ s'obtient alors grâce à un (seul) appel au solveur Partial-MSS (ligne 8).

Algorithme 6.1 : `Transformational-method` (calcul d'un consensus $\max_{\subseteq}$).

Input : Δ : une formule CNF satisfaisable (contraintes d'integrité);
$\mathcal{S} = \{\Phi_1, \ldots, \Phi_n\}$: un ensemble de formules CNF satisfaisables;
Admettons que les clauses de Φ_i sont désignées par $\alpha_i^1, \alpha_i^2, \ldots$;

Output : un consensus $max_{\subseteq}$ Γ de $\mathcal{S}$;

1 $\Sigma_1 \leftarrow \emptyset$; $\Sigma_2 \leftarrow \emptyset$;
2 $\Omega \leftarrow \bigcup_{\Phi_i \in \mathcal{S}} \{\neg\epsilon_i^j \vee \alpha_i^j$ s.t. $\alpha_i^j \in \Phi_i$ et où ϵ_i^j sont des variables fraîches$\}$;
3 $\Sigma_2 \leftarrow \{\epsilon_i^j\}_{i,j}$;
4 **foreach** $\Phi_i \in \mathcal{S}$ **do**
5 $\quad \Phi_i \leftarrow \Omega \cup \Phi_i \cup \Delta$;
6 $\quad$ Renommer toutes variables dans Φ_i (saut les ϵ_i^j) avec de nouvelles variables frâches;
7 $\quad \Sigma_1 \leftarrow \Sigma_1 \cup \Phi_i$;
8 $\Psi \leftarrow$ `Partial-MSS`(Σ_1, Σ_2);
9 $\Gamma \leftarrow \{\alpha_i^j \in \bigcup_{i=1}^n \Phi_i$ t.q. $\epsilon_i^j \in \Psi\}$;
10 **return** Γ;

Dans (Grégoire *et al.* 2016c), la viabilité de cette méthode est établie sur un large panel de benchmarks. Le principal facteur limitant cette méthode en pratique est la taille de l'instance transformée puisqu'on multiplie la taille de $\mathcal{S}$ par un facteur linéaire.

Méthode incrémentale

Le calcul d'un consensus $max_{\#}$ ne peut pas s'effectuer par une simple méthode incrémentale gloutonne. Par exemple, il n'est pas possible d'obtenir un $max_{\#}$ en procédant de la manière suivante : initialiser Γ à l'ensemble vide ; considérer successivement chaque clause α de $\bigcup_{i=1}^n \Phi_i$ et insérer α dans Γ si et seulement si $\Gamma \cup \{\alpha\} \cup \Delta \cup \Phi_i$ est satisfaisable pour chaque Φ_i. En effet, cet algorithme ne garantit pas que le consensus Γ soit maximal par rapport à la cardinalité car l'ensemble Γ à construire et la cardinalité de celui-ci peuvent dépendre de l'ordre dans lequel les clauses de $\bigcup_{i=1}^n \Phi_i$ sont considérées. En revanche, l'algorithme assure que Γ est un AC-MSS$(\bigcup_{i=1}^n \Phi_i, \mathcal{S})$ et donc un consensus $max_{\subseteq}$.

Nous avons adapté l'approche incrémentale, dépeinte dans l'algorithme 5.3, pour obtenir un AC-MSS Γ qui satisfait des contraintes d'intégrité Δ. D'abord, un pré-traitement peut être implémenté pour fournir

un sous-ensemble Γ qui est satisfaisable avec chaque $\Phi_i \cup \Delta$. Ensuite, Γ est étendu par un ensemble Ψ de clauses tel que $\Gamma \cup \Psi$ est satisfaisable avec chaque $\Phi_j \cup \Delta$.

6.3 Expérimentations

Dans cette section, nous décrivons l'étude expérimentale que nous avons menée dans le but de comparer les deux méthodes (incrémentale et par transformation) permettant de calculer un consensus $max_{\subseteq}$.

Benchmarks
Dans ces expérimentations, nous avons cherché à évaluer empiriquement les algorithmes sur un grand nombre d'instances qui sont réputées difficiles. Comme il n'y avait pas d'assez large panel d'instances spécifiquement dédié à la comparaison des outils de recherche de consensus, nous avons décidé de simuler un jeu d'instances pour le problème de recherche de consensus. Pour ce faire, nous avons utilisé des instances difficiles du problème SAT, ces dernières représentent des ensembles de clauses insatisfaisables qui peuvent être diviseés en séries de n sous-ensembles Φ_i satisfaisables. Nous avons considéré trois ensemble de benchmarks, à savoir *mus-sat11*, *ctt-itc7* et *ctt-rand*, issues de divers domaines d'application.

Le premier ensemble de benchmarks a été généré de la même manière que dans (Grégoire *et al.* 2016c). Nous avons considéré les 294 instances insatisfaisables provenant de la dernière compétition internationale de calcul de MUSes. [11] 8 ensembles d'instances ont été générés en considérant chaque nombre possible de sources $n \in \{5, 10, 15, 20, 25, 30, 35, 40\}$. Chaque instance du problème MUS est fractionné en n ensembles de clauses Φ_i ayant tous la même taille, et tel que chaque Φ_i soit satisfaisable. L'ensemble de benchmarks obtenu, noté *mus-sat11*, contient alors 2352 instances.

Les deux autres ensembles de benchmarks (*ctt-itc7* et *ctt-rand*) sont des instances du problème d'emploi du temps (*time-tabling problem*) décrit dans (Achá et Nieuwenhuis 2014), et plus précisément, du problème d'emploi du temps des cours (*curriculum-based course time-tabling problem*) défini dans la seconde compétition internationale de *timetabling* (ITC) tenue en 2007. [12] L'objectif dans ce problème est d'ordonner des séances de cours de différents modules sur la semaine étant donné un ensemble de salles et un ensemble de créneaux.

Étant donné un ensemble d'entités : jours, créneaux horaires, période, cours, enseignants, salles, bâtiments et cursus, le but est de trouver une affectation des cours aux salles et aux créneaux de sorte que un certain nombre de contraintes strictes (dures) qui relient ces entités, soient satisfaites avec un sous-ensemble maximal de contraintes souples. Les contraintes strictes interdisent que deux cours appartenant à un même cursus soient programmés à un même créneau. Aussi, elles interdisent que deux cours enseignés par le même enseignant aient lieu au même créneau. D'autres contraintes strictes expriment la disponibilité des enseignants et le nombre de séances qui doivent être programmées pour chaque cours. Les clauses correspondantes aux contraintes strictes sont considérées comme des contraintes d'intégrité dans la ré-interprétation de recherche de consensus de l'instance. D'autres critères à respecter sont considérés comme des contraintes souples. Ainsi, les séances de chaque cours doivent se dérouler sur un nombre donnée de jours, et les séances appartenant à un même cursus doivent se succéder, le nombre de séances par jour doit être compris entre un nombre minimum et un maximum de séances par jour. Les séances d'un même cours doivent se dérouler dans une même salle. De plus, certaines contraintes souples concernent la capacité des salles car, normalement, aucun cours ne doit être programmé dans une salle dont la capacité est inférieure au nombre d'étudiants inscrits dans ce cours. La recherche d'un consensus $max_{\subseteq}$ Γ dans ce contexte revient à calculer un sous-ensemble satisfaisable de clauses Γ de

11. http://www.satcompetition.org/2011/
12. http://www.cs.qub.ac.uk/itc2007/index_files/competitiontracks.htm

l'instance initiale tel que 1. Γ contient toutes les contraintes dures et tel que 2. Γ est satisfaisable avec chaque sous-ensemble Φ_i, tandis que l'ajout d'une toute autre clause de l'instance dans Γ violerait une de ces deux conditions.

ctt-itc7 contient 53 instances utilisées pour la compétition ITC. [13] Nous avons également enrichi ce jeu d'instances en considérant un ensemble supplémentaire de benchmarks *time-tabling*, noté *ctt-rand*, que nous avons généré de manière aléatoire, tout en fixant des valeurs réalistes pour certains paramètres. Les paramètres fixés étaient les suivants :

- nombre de salles : 12 ;
- nombre de bâtiments : 1 ;
- nombre de jours : 5 ;
- nombre de périodes : 4 ;
- nombre de cours pour chaque enseignant : 4.

Les autres paramètres varient comme suit :

- nombre de cours (60, 46) ;
- nombre de cursus (26, 30, 43) ;
- nombre de cours par cursus [2,. . . ,5] ;
- capacité de chaque salle [9,. . . ,300] ;
- nombre d'étudiants [10,. . . ,120] ;
- nombre de séances par jours [2,. . . ,5] ;
- pour chaque salle, la probabilité que la salle adaptée au cours soit sélectionnée est entre $\{30\%, 50\%, 70\%, 80\%\}$;
- pour chaque cours et chaque période, la probabilité que ce cours puisse être programmé pendant cette période varie entre les trois valeurs 20%, 80% and 90%.

Nous avons implémenté un générateur de benchmarks *timetabling*, encodés en XML, suivant le format défini dans la compétition PATAT. [14] Nous avons ainsi généré six ensembles d'instances *timetabling* de 25 cours, selon les paramètres décrits avant. Ainsi, *ctt-rand* est constitué au total de 150 instances, tel que : le nombre de cours et de cursus sont respectivement, 30 et 14 pour le premier ensemble, 46 et 26 pour le deuxième ensemble, 46 et 30 pour le troisième et quatrième ensemble, 60 et 40 pour les deux derniers ensembles.

Tous ces benchmarks ont été vérifiés avec le solveur weighted Partial-MAX-SAT `MaxHS` [15] (Davies et Bacchus 2011; 2013b;a) dans le but de s'assurer que dans chaque instance il existe au moins deux sources Φ_i et Φ_j qui sont mutuellement contradictoires.

Hardware, software et time-out

Nous avons mené nos expérimentations sur des machines équipées de processeurs Intel Xeon E5-2643 (3.30GHz), de 64 Gb de mémoire vive et du système d'exploitation Linux CentOS. Nous avons considéré un *time-out* de 900 secondes et fixé le *memory-out* à 8 Gb pour chaque instance testée. Nous avons utilisé l'outil `CMP` de (Grégoire *et al.* 2014a) pour le calcul d'un MSS. Nous avons implanté les deux algorithmes `Transformational method` et `Incremental-method` en C++, directement sur le solveur SAT GLUCOSE (Audemard et Simon 2009) (`http://www.labri.fr/perso/lsimon/glucose`).

13. `http://tabu.diegm.uniud.it/ctt/`
14. `http://www.patatconference.org`
15. `http://www.maxhs.org`

Résultats

Les figures 6.1 et 6.2 reportent les résultats expérimentaux pour les benchmarks *mus-sat11*. La figure 6.1 présente une comparaison entre `Incremental-method` et `Transformational-method` sur les temps de résolution exprimés en secondes (CPU), et la figure 6.2 reporte le nombre total d'instances résolues par chacune de ces deux méthodes. Comme nous pouvons le voir, `Incremental-method` affiche de meilleures performances, à la fois en ce qui concerne les temps de résolution et le nombre d'instances résolues.

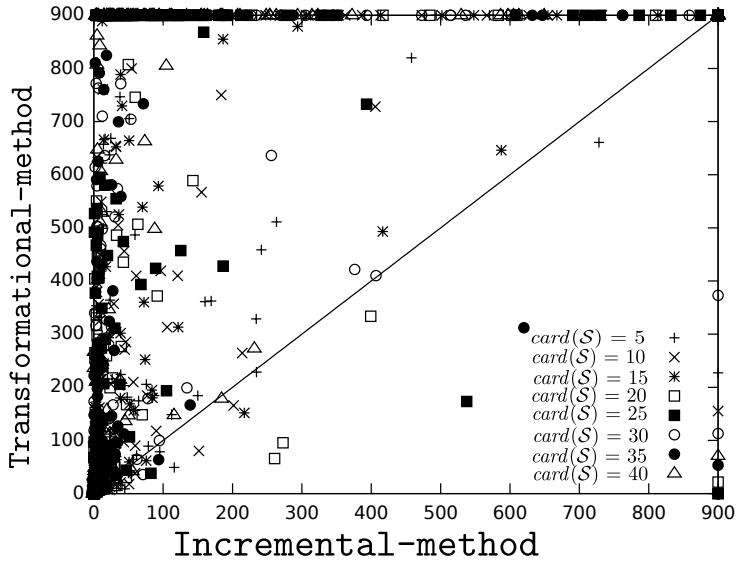

FIGURE 6.1 – Temps d'exécution de `Incremental-method` vs. `Transformational-method` sur les benchmarks *mus-sat11*.

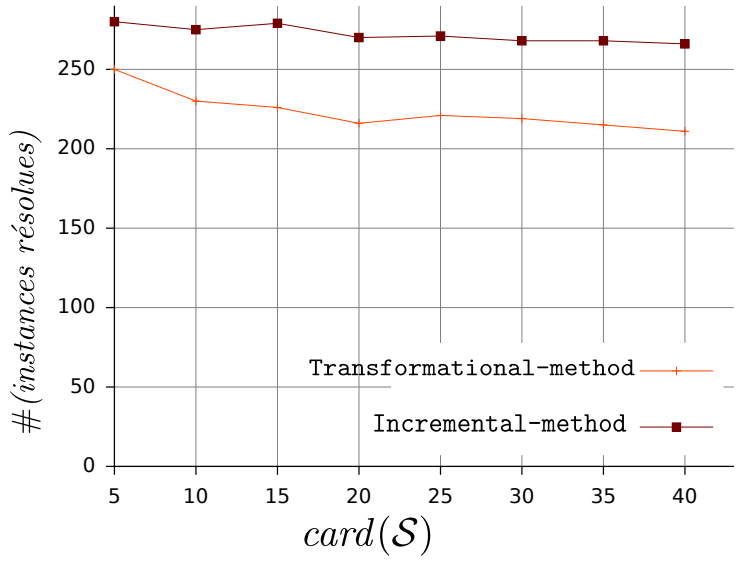

FIGURE 6.2 – Nombre d'instances résolues pour les benchmarks *mus-sat11*.

La table 6.1 reporte les résultats expérimentaux sur les benchmarks *ctt-itc7*. La première colonne affiche le nom des instances. Les trois autres colonnes suivantes reportent des informations sur l'instance du benchmark : *#vars* désigne le nombre de variables ; *#clauses* représente le nombre de clauses et $\#\Phi_i$

est le nombre de sources dans l'instance. Les deux derniers groupes de colonnes reportent respectivement les résultats obtenus par `Transformational-method` et `Incremental-method`. *status* affiche " OK" lorsque l'instance a été résolue ; "MO" lorsque la limite de mémoire autorisée est dépassée ; "TO" lorsque la limite de temps de résolution est dépassée. *time* indique le temps de résolution, exprimé en secondes (CPU), lorsque un consensus $max_{\subseteq}$ Γ est retourné par la méthode. La dernière colonne (#Γ), reporte la taille du consensus calculé, c'est-à-dire le nombre de clauses dans le consensus. La table montre que `Incremental-method` trouve un consensus $max_{\subseteq}$ pour chaque instance (constamment en moins de 136 secondes) tandis que `Transformational-method` échoue dans 49 instances parmi les 53, soit à cause du dépassement de la limite du *time-out* ou bien du *memory-out*.

La table 6.2 reporte un échantillon des résultats expérimentaux sur les benchmarks *ctt-rand*. Comme tous les résultats étaient assez similaires, nous avons choisi de mettre dans la table 10 ensembles d'instances sélectionnées au hasard, en fonction des valeurs assignées aux paramètres : (*#courses_#curricula_%slot_%room-suitability*). La table montre que `Incremental-method` a résolu toutes les instances de *ctt-rand* en moins de 2 secondes. De plus, `Transformational-method` n'arrivait pas à trouver un consensus $max_{\subseteq}$ sur 20/40 instances et il a fallu plus de temps pour extraire un consensus $max_{\subseteq}$ dans chacune des 20 instances résolues.

À travers ces résultats expérimentaux, il est clair que `Incremental-method` est expérimentalement plus efficace que `Transformational-method` sur nos différents benchmarks. De plus `Transformational-method` rencontre des dépassements de mémoire sur un grand nombre d'instances car cette dernière utilise une représentation qui multiplie la taille de l'instance au moins par un facteur $\mathcal{O}(n)$.

6.4 Conclusion

Cette étude complète quelques résultats récents (Grégoire *et al.* 2016c) sur la recherche de consensus dans la logique propositionnelle. (Grégoire *et al.* 2016c) ont proposé une méthode générique, dite par transformation, pour le recherche d'un consensus en fonction de divers critères de maximalité, y compris la maximalité par rapport à l'inclusion ensembliste. Nous avons proposé une méthode incrémentale spécifique pour le calcul d'un consensus maximal par rapport à l'inclusion ensembliste. Cette méthode permet d'éviter les éventuelles explosions de l'espace mémoire de la méthode par transformation et a été prouvée expérimentalement plus efficace sur tous les benchmarks testés.

Nom de l'instance	Données du benchmark			`Transformational-method`			`Incremental-method`		
	#vars	#clauses	#Φ_i	statut	temps	#Γ	statut	temps	#Γ
comp01	4950	93170	38	OK	29.11	1104	OK	0.04	1105
comp02	23016	1421664	41	MO	?	?	OK	2.95	12777
comp03	20536	1104622	29	MO	?	?	OK	2.28	11297
comp04	23706	1470609	27	MO	?	?	OK	2.30	9402
comp05	22914	591566	86	MO	?	?	OK	6.73	19159
comp06	32012	2722482	57	MO	?	?	OK	6.15	14409
comp07	41840	4416587	76	MO	?	?	OK	8.21	17202
comp08	25754	1734683	37	MO	?	?	OK	2.97	9769
comp09	23814	1372119	43	MO	?	?	OK	2.76	11637
comp10	33710	3077495	55	MO	?	?	OK	6.33	14235
comp11	7320	120548	37	OK	144.99	1308	OK	0.07	1309
comp12	33768	1685796	24	MO	?	?	OK	14.81	22598
comp13	26014	1671845	43	MO	?	?	OK	2.69	11022
comp14	24335	1610297	28	MO	?	?	OK	2.84	11091
comp15	20536	1104494	29	MO	?	?	OK	2.14	11242
comp16	34870	3022190	60	MO	?	?	OK	4.55	14873
comp17	28349	2168040	50	MO	?	?	OK	4.88	12452
comp18	14977	402252	99	TO	?	?	OK	1.76	6104
comp19	20912	1156777	32	MO	?	?	OK	1.82	9666
comp20	37417	3594812	73	MO	?	?	OK	7.57	17342
comp21	28716	2079155	54	MO	?	?	OK	4.13	13875
DDS1	123591	2235240	44	MO	?	?	OK	135.97	51573
DDS2	35646	2585114	59	MO	?	?	OK	10.15	7944
DDS3	13090	583659	48	TO	?	?	OK	0.87	2432
DDS4	123340	6842064	46	MO	?	?	OK	59.40	39813
DDS5	61272	7801619	23	MO	?	?	OK	10.74	10929
DDS6	29957	2520532	44	MO	?	?	OK	5.74	12082
DDS7	17327	701224	80	TO	?	?	OK	1.73	5220
EA02	17968	1029158	65	TO	?	?	OK	1.75	7444
EA03	157225	8274241	61	MO	?	?	OK	46.38	52134
EA04	71054	4918070	25	MO	?	?	OK	10.97	13044
EA05	58758	8013557	99	MO	?	?	OK	13.22	12272
EA06	17600	984792	63	TO	?	?	OK	1.08	5271
EA07	136162	2813432	66	MO	?	?	OK	72.98	57807
EA08	39569	5389918	15	MO	?	?	OK	5.06	5800
EA09	81074	0981249	79	MO	?	?	OK	20.80	31091
EA10	26268	808058	90	TO	?	?	OK	5.43	8070
EA11	17532	1186698	70	MO	?	?	OK	1.68	10572
EA12	25956	2304322	77	MO	?	?	OK	3.22	9814
test1	9734	274988	65	OK	402.99	2262	OK	0.26	2263
test2	11028	349278	79	TO	?	?	OK	0.38	2897
test3	13400	441166	08	TO	?	?	OK	0.82	4955
test4	12210	408077	06	TO	?	?	OK	0.92	4376
toy	488	1606	6	OK	0.00	86	OK	0.00	87
Udine1	47176	5470432	89	MO	?	?	OK	14.30	26646
Udine2	50156	6219082	95	MO	?	?	OK	14.93	19996
Udine3	29957	2520130	44	MO	?	?	OK	4.31	11881
Udine4	17506	817156	12	MO	?	?	OK	1.36	7373
Udine5	46423	5371246	80	MO	?	?	OK	12.66	20613
Udine6	41181	4358192	54	MO	?	?	OK	6.97	15001
Udine7	43780	4776206	59	MO	?	?	OK	6.39	15068
Udine8	51110	5799551	11	MO	?	?	OK	12.60	22638
Udine9	40516	4036770	69	MO	?	?	OK	7.88	18440

TABLE 6.1 – Résultats expérimentaux sur les benchmarks *ctt-itc7*.

Nom de l'instance	Données du benchmark			`Transformational-method`			`Incremental-method`		
#courses_#curricula_%slot_%room	#vars	#clauses	$\#\Phi_i$	statut	temps	$\#\Gamma$	statut	temps	$\#\Gamma$
random_30_14_90_80_111	4910	85988	22	OK	41.36	1655	OK	0.18	1632
random_30_14_90_80_122	4910	85100	22	OK	28.04	1468	OK	0.08	1467
random_30_14_90_80_146	4910	84696	22	OK	26.73	1438	OK	0.18	1441
random_30_14_90_80_170	4910	85044	22	OK	19.29	1464	OK	0.08	1463
random_30_14_90_80_205	4910	86376	22	OK	28.77	1654	OK	0.18	1647
random_30_14_90_80_246	4910	85962	22	OK	39.92	1664	OK	0.11	1666
random_30_14_90_80_277	4910	85484	22	OK	29.73	1544	OK	0.15	1550
random_30_14_90_80_285	4910	85376	22	OK	29.38	1586	OK	0.18	1577
random_30_14_90_80_300	4910	86256	22	OK	26.67	1722	OK	0.13	1699
random_30_14_90_80_311	4910	85610	22	OK	28.01	1594	OK	0.07	1605
random_46_26_20_30_127	9734	276372	37	OK	454.74	2268	OK	0.34	2268
random_46_26_20_30_128	9734	276058	38	OK	320.11	2171	OK	0.26	2171
random_46_26_20_30_144	9734	275710	38	OK	392.68	2190	OK	0.36	2190
random_46_26_20_30_209	9734	276764	38	OK	411.95	2424	OK	0.26	2424
random_46_26_20_30_216	9734	276032	38	OK	341.69	2235	OK	0.31	2235
random_46_26_20_30_248	9734	274840	38	OK	202.25	1910	OK	0.36	1910
random_46_26_20_30_24	9734	275218	38	OK	194.99	2023	OK	0.36	2023
random_46_26_20_30_255	9734	275812	38	OK	337.22	2169	OK	0.36	2169
random_46_26_20_30_270	9734	276624	38	OK	486.83	2395	OK	0.37	2395
random_46_26_20_30_275	9734	275872	38	OK	222.05	2159	OK	0.16	2159
random_46_30_80_50_117	9894	283010	41	TO	?	?	OK	0.66	3697
random_46_30_80_50_13	9894	283536	41	TO	?	?	OK	0.66	3968
random_46_30_80_50_146	9894	281354	42	TO	?	?	OK	0.62	3570
random_46_30_80_50_155	9894	283598	42	TO	?	?	OK	0.76	3944
random_46_30_80_50_191	9894	283840	42	TO	?	?	OK	0.76	3929
random_46_30_80_50_220	9894	281940	42	TO	?	?	OK	0.73	3606
random_46_30_80_50_238	9894	283442	42	TO	?	?	OK	0.76	3844
random_46_30_80_50_28	9894	282962	42	TO	?	?	OK	0.63	3958
random_46_30_80_50_292	9894	284068	42	TO	?	?	OK	0.61	3915
random_46_30_80_50_308	9894	281926	42	TO	?	?	OK	0.66	3611
random_60_40_80_70_107	15580	623240	53	TO	?	?	OK	1.33	6431
random_60_40_80_70_108	15580	625546	53	TO	?	?	OK	1.33	6975
random_60_40_80_70_110	15580	625318	52	TO	?	?	OK	1.43	6945
random_60_40_80_70_120	15580	625682	53	TO	?	?	OK	1.43	7186
random_60_40_80_70_12	15580	624516	53	TO	?	?	OK	1.33	6779
random_60_40_80_70_201	15580	623410	53	TO	?	?	OK	1.23	6660
random_60_40_80_70_203	15580	625576	53	TO	?	?	OK	1.73	6888
random_60_40_80_70_258	15580	624272	53	TO	?	?	OK	1.43	6355
random_60_40_80_70_275	15580	624072	53	TO	?	?	OK	1.42	6660
random_60_40_80_70_281	15580	622790	53	TO	?	?	OK	1.33	6304

TABLE 6.2 – Quelques résultats expérimentaux sur les benchmarks *ctt-rand*.

Chapitre

7

Consensus admissibles

Sommaire

Les consensus peuvent être définis de différentes manières et obéissent à une variété de propriétés. Cette étude étend l'approche définissant une forme de consensus dans un cadre de logique standard introduite dans (Grégoire *et al.* 2016c, Grégoire et Lagniez 2016b) et qui s'applique à des sources d'informations, positions, plans, désirs, objectifs, connaissances et croyances encodées à l'aide de formules propositionnelles. L'approche caractérise et calcule des formes de consensus maximaux qui vont au-delà des informations partagées par chaque agent. En effet, elle exprime une attitude très libérale où chaque agent accepte les informations provenant d'un autre agent dans la mesure où il n'est pas contredit par ces nouvelles informations et dans la mesure où l'information globale acceptée reste non contradictoire. En même temps, l'agent accepte de laisser tomber des fragments de sa base d'informations qui contredisent un autre agent. Ainsi, un consensus maximal devrait capturer autant d'informations que possible incluses dans l'ensemble des sources d'informations exprimées par les agents, sans devenir incohérent et sans entrer en conflit avec aucun des agents. Par conséquent, un consensus peut contenir certaines informations qu'un agent donné ne possède pas et qu'il n'est même pas capable de déduire. Cependant, cet agent pourrait être conduit à approuver le consensus car il n'est logiquement contredit ni par les informations contenues dans le consensus, ni par les conséquences logiques pouvant être déduites de ce consensus et ni de ses propres informations. Lorsque les informations traduisent les désirs contradictoires de différents agents, un consensus maximal contiendra autant de désirs que possible sans contredire n'importe quel désir de n'importe quel agent. Dans le contexte de la logique propositionnelle, le concept de consensus maximal a donc été défini par (Grégoire *et al.* 2016c) comme étant un sous-ensemble maximal satisfaisable, où la maximalité est par rapport à la cardinalité ou l'inclusion ensembliste, de toutes les formules telles que ce sous-ensemble est également satisfaisable avec chaque agent ou source d'informations considéré individuellement.

Dans cette étude, ce concept de consensus est caractérisé par une notion d'admissibilité, en filtrant certains consensus qui ne peuvent être approuvés par les agents, en raison de la représentation logique *ad-hoc* des connaissances et paradoxes liés à l'implication matérielle. Une approche de calcul d'un consensus maximal admissible est donc introduite.

Le chapitre est organisé comme suit. Dans la section suivante, nous décrivons de manière intuitive les circonstances spécifiques dans lesquelles certains consensus maximaux de (Grégoire *et al.* 2016c) peuvent figurer indésirables par certains agents. Dans les sections 7.2 et 7.3, des consensus admissibles sont introduits. Les consensus maximaux admissibles et admissibles maximaux sont définis et comparés

dans la section 7.4. Les sections 7.5 et 7.6 décrivent comment les consensus admissibles maximaux et maximaux admissibles peuvent être obtenus en utilisant une extension des mécanismes de (Grégoire *et al.* 2016c). Enfin, une discussion sur les voies prometteuses pour des travaux futurs est présentée.

7.1 Consensus indésirables

Afin de mieux expliquer la motivation de cette étude, nous allons d'abord donner un exemple intuitif d'un consensus maximal indésirable. Supposons que nous devons trouver un consensus maximal entre deux agents qui ne sont pas mutuellement logiquement contradictoires et sont représentés par leurs propres sources d'informations, notées respectivement Φ_1 and Φ_2. En l'absence de contradiction logique entre les agents, l'unique consensus maximal, tel que défini dans (Grégoire *et al.* 2016c), est $\Phi_1 \cup \Phi_2$. En effet, cette approche spécifique du consensus attend de chaque agent qu'il soit aussi accommodant que possible, dans la mesure où aucune contradiction ne se présente : d'un point de vue logique, un sous-ensemble maximal de la base de connaissance $\Phi_1 \cup \Phi_2$ à extraire doit être satisfaisable et ne contredire aucun agent. Bien que $\Phi_1 \cup \Phi_2$ représente l'unique consensus maximal, lorsque cet ensemble est satisfaisable, dans certains cas, il pourrait ne pas être approuvé par tous les agents, même dans cette attitude libérale spécifique.

Supposons, par exemple, que le premier agent affirme la règle « *Si le prisonnier est coupable, il mérite la prison* » tandis que le second agent affirme « *Le prisonnier n'est pas coupable* » et « *Le prisonnier mérite la prison* ». Supposons également que cela s'exprime par les ensembles de formules propositionnelles $\Phi_1 = \{\textit{coupable} \rightarrow \textit{mérite-prison}\}$ et $\Phi_2 = \{\neg\textit{coupable}, \textit{mérite-prison}\}$. Soulignons que Φ_1 et Φ_2 ne sont pas logiquement contradictoires. De plus, l'ensemble $\Phi_1 \cup \Phi_2$ est satisfaisable avec Φ_1 et avec Φ_2 : il représente donc l'unique consensus maximal. Pourtant, le premier agent pourrait ne pas approuver $\Phi_1 \cup \Phi_2$ comme consensus. En effet, en affirmant la règle « *Si le prisonnier est coupable, il mérite la prison* », il aurait peut être voulu rejeter la situation où le prisonnier n'est pas coupable mais mérite la prison. Si c'est le cas, on peut prétendre que ce problème se produit parce que le premier agent a encodé ses informations de manière incorrecte : il aurait dû écrire « *Si le prisonnier mérite la prison alors il est coupable* », ou si il ne voit pas d'autres possibilités de mériter la prison : « *Le prisonnier mérite la prison si et seulement si il est coupable* ». Dans les deux cas $\Phi_1 \cup \Phi_2$ devient insatisfaisable et ce n'est plus un consensus ; en fait, aucun consensus non vide existe entre les deux agents, ce qui correspond au résultat attendu.

Dans cet exemple, on peut être tenté de conclure qu'un consensus maximal est rejeté par un agent parce que l'encodage logique est incorrect dans le sens où ceci ne représente pas ce que cet agent avait en tête. Cependant, cette attitude ne nous aide pas à construire des systèmes d'intelligence artificielle dont le comportement et les conclusions doivent répondre aux attentes des agents. En conséquence, nous ne justifierons pas par "ceci est la faute de l'utilisateur" pour des consensus maximaux inattendus. Au contraire, nous considérons que ce genre d'encodage logique maladroit des informations arrive fréquemment et qu'il faut faire avec. Par conséquent, un système d'intelligence artificielle doit automatiquement éviter les consensus que certains agents n'approuveraient probablement pas. Il devrait d'abord se focaliser sur des consensus qui ne sont pas litigieux. Nous acceptons donc que les gens encodent parfois $A \rightarrow B$ quand ils pensent effectivement $B \rightarrow A$ ou même $A \equiv B$. À cet égard, cette étude ne porte donc pas sur la manière correcte dont les agents devraient encoder les connaissances en logique propositionnelle. Au contraire, nous détectons des consensus qui pourraient ne pas être approuvés par certains agents selon leur manière habituelle de représenter les connaissances. Bien évidemment, notre approche de recherche de consensus devrait également tenir compte de la possibilité que l'agent ait utilisé les implications matérielles de manière correcte.

Soulignons également que nous adoptons le langage de la logique propositionnelle car il est le plus facile à apprendre et à utiliser par les non-logiciens, parmi les logiques pour lesquelles des vérificateurs de satisfiabilité expérimentalement efficaces sont largement disponibles. En effet, comme nous le verrons, ces vérificateurs s'avèrent très utiles pour mettre en œuvre un mécanisme de recherche de consensus maximal. À l'heure actuelle, des vérificateurs efficaces ne sont pas disponibles pour les logiques avec un connecteur d'implication plus "naturel", comme la *logique conditionnelle* (voir par exemple (Edgington 2008, Chellas 1975) pour en savoir plus sur les logiques conditionnelles).

Passons à l'analyse et l'explication plus en détail de notre motivation. Considérons à nouveau notre exemple des deux agents présenté précédemment. Tout d'abord, notons que si Φ_1 était « $\{$*mérite-prison* $\rightarrow$ *coupable*$\}$ » alors Φ_1 ne serait pas en mesure de dériver *mérite-prison* si il savait aussi que *coupable* est *vrai*. L'incapacité d'inférer *mérite-prison* dans ce cas pourrait être une raison motivante pour écrire plutôt « $\{$*coupable* $\rightarrow$ *mérite-prison*$\}$ ».

Les logiciens voient souvent des personnes sans connaissances solides en logique représenter des formules avec des connecteurs d'implication de manière inversée, comme l'illustre l'exemple ci-dessus. Une explication possible à cela pourrait donc être une préférence pour l'expression des règles de décision dont le rôle prévu est de dériver les conséquences lorsque leurs antécédents sont *vrais*. Cela est aussi parfois dû à une tendance naturelle à rester proche des mécanismes de chaînage causal qui pourraient impliquer des étapes successives. En conséquence, ils ne perçoivent pas qu'ils optent pour un encodage logique erroné en ce qui concerne l'implication matérielle de la logique classique. Plus généralement, ils ont souvent tendance à supposer qu'une règle de la forme *Si A alors B* est prise en compte par un système d'intelligence artificielle que lorsque A est *vrai* (et parfois, par contraposition lorsque B est *faux*). Comme dans cet exemple, ils ne s'attendent pas à ce que la règle couvre une situation où $\neg A$ et B sont simultanément *vrais*.

Il est crucial de souligner que dans les négociations difficiles impliquant un grand nombre de formules et d'informations contradictoires, il peut y avoir un nombre exponentiel de consensus maximaux. L'énumération des consensus est souvent une tâche hors de portée dans ces circonstances. Néanmoins, dans ces situations, l'extraction d'un consensus maximal est souvent un résultat satisfaisant pour le groupe d'agents. Cela peut être la solution finale ou un point de départ intéressant pour la poursuite des négociations. Ainsi, l'approche que nous proposons vise à éviter les consensus maximaux qui pourraient ne pas convenir aux agents. Cependant, si aucun autre consensus maximal ne peut être trouvé au bout d'une certaine limite de temps, nous extrayons alors les consensus maximaux éventuellement problématiques ou recherchons des consensus qui ne sont pas maximaux.

En récapitulant le tout, la question abordée dans cette étude est la caractérisation du complément d'une catégorie de consensus maximaux qui pourraient ne pas être approuvés par les agents concernés, parce que certains de ces agents pourraient avoir exprimé leurs informations en utilisant le plus intuitif – mais incorrect – encodage d'informations dans la logique classique. Cela concerne également l'introduction d'une technique permettant pour un système d'intelligence artificielle de proposer ces consensus possiblement problématiques. En d'autres termes, nous introduisons une nouvelle forme de préférence dans la recherche de consensus qui évite les consensus qui reposent éventuellement sur une mauvaise compréhension commune de la logique standard par les agents.

7.2 Consensus admissibles

À partir de maintenant, nous considérons les règles de la forme $\gamma \rightarrow b$ où γ est une conjonction (ou, de manière équivalente, un ensemble) de littéraux et b est un littéral. Nous supposons qu'il n'y a

pas de règles circulaires, c'est-à-dire que nous considérons que $b \notin \gamma$ et $\neg b \notin \gamma$. Nous supposons que $\Phi_i \in [\Phi_1, \ldots, \Phi_n]$ et Γ désigne un consensus de $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$. Nous notons $\{\gamma_j \rightarrow b\}_i$ l'ensemble des règles dans Φ_i ayants b comme conséquence ; $\bigwedge_j \neg\gamma_j$ représente la conjonction de la négation de chaque γ_j de cet ensemble de règles. $Cn(\Phi_i)$ représente l'ensemble des conséquences déductives (aussi appelé la fermeture déductive) de Φ_i.

L'ensemble des formules $\{\gamma \rightarrow b\}$ est satisfaisable avec $\{\neg\gamma, b\}$; $\gamma \rightarrow b$ est à la fois une conséquence déductive de $\neg\gamma$ et une conséquence déductive de b. Un agent qui affirme la règle $\gamma \rightarrow b$ lorsque γ est *faux*, est en situation contrefactuelle. En effet, pour une règle dont l'antécédent est *faux*, elle ne peut pas être "admise" dans le sense où sa conséquence ne peut être inférée parce que l'antécédent n'est pas vrai (ne tient pas). Lorsque la règle est exprimée en formule propositionnelle et quand γ est à *faux*, la règle ne peut que être interprétée comme étant *vrai* quelle que soit sa conséquence et la valeur de vérité de cette conséquence. Ainsi, γ et b peuvent être interprétés comme étant respectivement *faux* et *vrai*, tandis que la règle considérée dans son ensemble est alors *vrai*.

Cependant, comme décrit dans les sections précédentes, dans certaines circonstances spécifiques, lorsqu'un agent Φ_i affirme une règle $\gamma \rightarrow b$, il pourrait ne pas envisager et supposer que b et $\neg\gamma$ soient simultanément à *vrai*. Par conséquent, dans ces circonstances qui restent à décrire, il n'approuvera aucun consensus Γ tel que $\Gamma \cup \Phi_i \models b \wedge \neg\gamma$.

Nous allons maintenant dresser la motivation de cette proposition de manière intuitive. Tout d'abord, notons que nous considérons $b \wedge \neg\gamma$ comme une conséquence déductive de $\Gamma \cup \Phi_i$ (vs. de Φ_i) dans cette proposition. En effet, rappelons que toutes les formules de Φ_i sont prises en compte dans la définition du consensus ; de même, nous supposons que chaque agent prend également en compte toute sa base d'informations Φ_i en plus de Γ afin d'évaluer si oui ou non il approuve Γ. Maintenant, le schéma du raisonnement contrefactuel que nous avons décrit, se produit lorsque $\Gamma \cup \Phi_i \models b \wedge \neg\gamma$. La question que nous devons aborder est dans quelles circonstances l'agent qui a exprimé Φ_i pourrait rejeter un tel consensus Γ. Ces circonstances arrivent lorsque $\Phi_i \models \gamma \rightarrow b$ avec des conditions supplémentaires que nous allons donc explorer.

Nous supposons que ces conditions sont liées aux différents ensembles possibles de modèles partiels de Φ_i qui fixent les valeurs de vérité de b et γ. En conséquence, nous explorerons dans la section suivante toutes les valeurs de vérité possibles pour b et γ dans les modèles de Φ_i afin de détecter les situations où $\Phi_i \models \gamma \rightarrow b$ et $\Gamma \cup \Phi_i \models b \wedge \neg\gamma$ peuvent réellement avoir lieu. Dans chacune de ces situations, nous discutons si l'agent correspondant pourrait rejeter Γ en raison d'un schéma de raisonnement contrefactuel non envisagé. Cette analyse nous conduit à la définition suivante de consensus *admissible*.

Définition 7.1. Consensus admissible
Un **consensus admissible** Γ de $\mathcal{S}$ est un consensus pour $\mathcal{S}$ tel que pour tout littéral b, on a pour tout ensemble non-vide $\{\gamma_j \rightarrow b\}_i$, $\Gamma \cup \Phi_i \not\models b \wedge \bigwedge_j \neg\gamma_j$ chaque fois que, en même temps, $\Phi_i \not\models b$ et $\Phi_i \not\models \neg b$.

Nous donnerons une justification détaillée pour cette définition dans la section suivante. Pour le moment, commençons par donner quelques propriétés de base des consensus admissibles.

Une position naturellement attendue d'un agent qui a exprimé Φ_i est l'approbation de tout consensus qui est un sous-ensemble de ses propres informations Φ_i avec les conséquences déductives de Φ_i, c'est-à-dire $Cn(\Phi_i)$. Il est facile de constater qu'un tel consensus est admissible.

Propriété 7.1. Tout consensus Γ de $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$ tel que $\Gamma \subseteq Cn(\Phi_i)$ est admissible.

En effet, quand $\Gamma \subseteq Cn(\Phi_i)$, on a $Cn(\Gamma \cup \Phi_i) \subseteq Cn(\Phi_i)$ et Γ est non admissible seulement lorsque, au même temps, $\Phi_i \not\models b$ et $\Phi_i \cup \Gamma \models b$, ce qui contredit $Cn(\Gamma \cup \Phi_i) \subseteq Cn(\Phi_i)$.

Les consensus admissibles existent toujours. Comme conséquence directe de la définition 7.1, on a que l'ensemble vide est toujours un consensus admissible. Rappelons que n'importe quel profil $\mathcal{S}$ est constitué d'ensembles de formules où chaque ensemble est satisfaisable.

Propriété 7.2. Pour tout profil $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$, il existe au moins un consensus admissible pour $\mathcal{S}$.

Enfin, notons qu'un agent qui exprime la règle $\gamma \rightarrow b$ dans Φ_i et qui n'envisage pas la possibilité d'avoir b à *vrai* et γ à *faux*, pourrait toutefois approuver un consensus Γ tel que $\Gamma \models b$ et $\Gamma \not\models \gamma$ quand Φ_i ne permet pas de déduire $\neg b$. Bien qu'un tel Γ semble être en contradiction avec la volonté implicite de l'agent qui a exprimé Φ_i, en réalité ce Γ ne peut avoir lieu que quand $\Phi_i \cup \Gamma \models \gamma$. Par conséquent, Γ pourrait être admissible ; encore une fois, cela veut dire que Γ est considéré conjointement avec Φ_i par cet agent.

À cet égard, un agent qui accepte un consensus est autorisé à explorer les conséquences dérivées à partir de l'union du consensus et ses propres objectifs ou informations, c'est-à-dire les conséquences de $\Phi_i \cup \Gamma$. Dans une certaine mesure, pour certains domaines d'application, l'approbation de Γ par un agent pourrait traduire une attitude hypocrite puisque l'agent utilise ses propres informations pour évaluer le consensus. Il pourrait donc être conscient de certaines conséquences du consensus et ne pas avoir besoin de partager cela avec les autres agents qui eux ne peuvent pas envisager ces conséquences. Par exemple, un groupe d'agents peut s'entendre sur un consensus alors que certains parmi eux savent en réalité que l'adoption du consensus posera des problèmes aux autres agents qui ne disposent pas des informations nécessaires pour envisager ces problèmes. Maintenant, il est facile de voir qu'un tel Γ "*hypocrite*" n'est pas exclu par le concept d'admissibilité, dans la mesure où les conditions décrites dans la définition 7.1 concernant b and γ (et leurs négations) sont respectées. Notons que toute formule de $\mathcal{S}$ doit être prise en compte dans le calcul des consensus ; aucune information ne peut être dissimulée. Cependant, cela ne nécessite pas que ces informations doivent être partagées avec chaque agent.

Mettons l'accent sur le fait que les consensus non admissibles ne seront pas écartés dans l'approche de calcul de consensus que nous allons présenter : simplement, un ordre de préférence est établi sur les consensus, en ce sens les consensus admissibles ou maximaux admissibles seront calculés en premier. Lorsque ces derniers n'existent pas ou ne sont pas trouvés, les consensus non admissibles, peuvent alors être calculés et suggérés aux agents.

7.3 Analyse détaillée

Dans cette section, nous donnons la justification formelle et détaillée qui nous a mené à la définition 7.1. Nous analysons les différentes situations possibles où $\Phi_i \models \gamma \rightarrow b$ et $\Gamma \cup \Phi_i \models b \wedge \neg\gamma$. Evidemment, ces deux conditions impliquent $\Gamma \cup \Phi_i \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$. Nous allons effectuer un raisonner au cas par cas : l'arbre de décision correspondant est fourni dans la figure 7.1.

D'abord, supposons que Φ_i contient la règle $\gamma \rightarrow b$. Nous discuterons du cas où nous avons $\Phi_i \models \gamma \rightarrow b$ et $\gamma \rightarrow b \notin \Phi_i$, plus tard. Notons que la situation où $\Phi_i \not\models \gamma \rightarrow b$ traduit l'absence de raisonnement contrefactuel de Φ_i que nous voulons détecter ; une telle situation n'empêche pas $\Gamma \cup \Phi_i \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$ de se produire, mais dans cette étude, cela ne devrait pas poser de problème à l'agent correspondant dans la mesure où il est d'accord avec les principes sous-jacents au concept de consensus et les suit.

1. Quand $\Phi_i \models \gamma$, $\nexists\, \Gamma$ tel que $\Gamma \models \neg\gamma$ par définition d'un consensus. Donc, $\nexists\, \Gamma$ tel que $\Gamma \cup \Phi_i \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$. Notons aussi que $\nexists\, \Gamma$ tel que $\Gamma \models \neg b$ puisqu'un tel Γ serait en contradiction avec Φ_i.

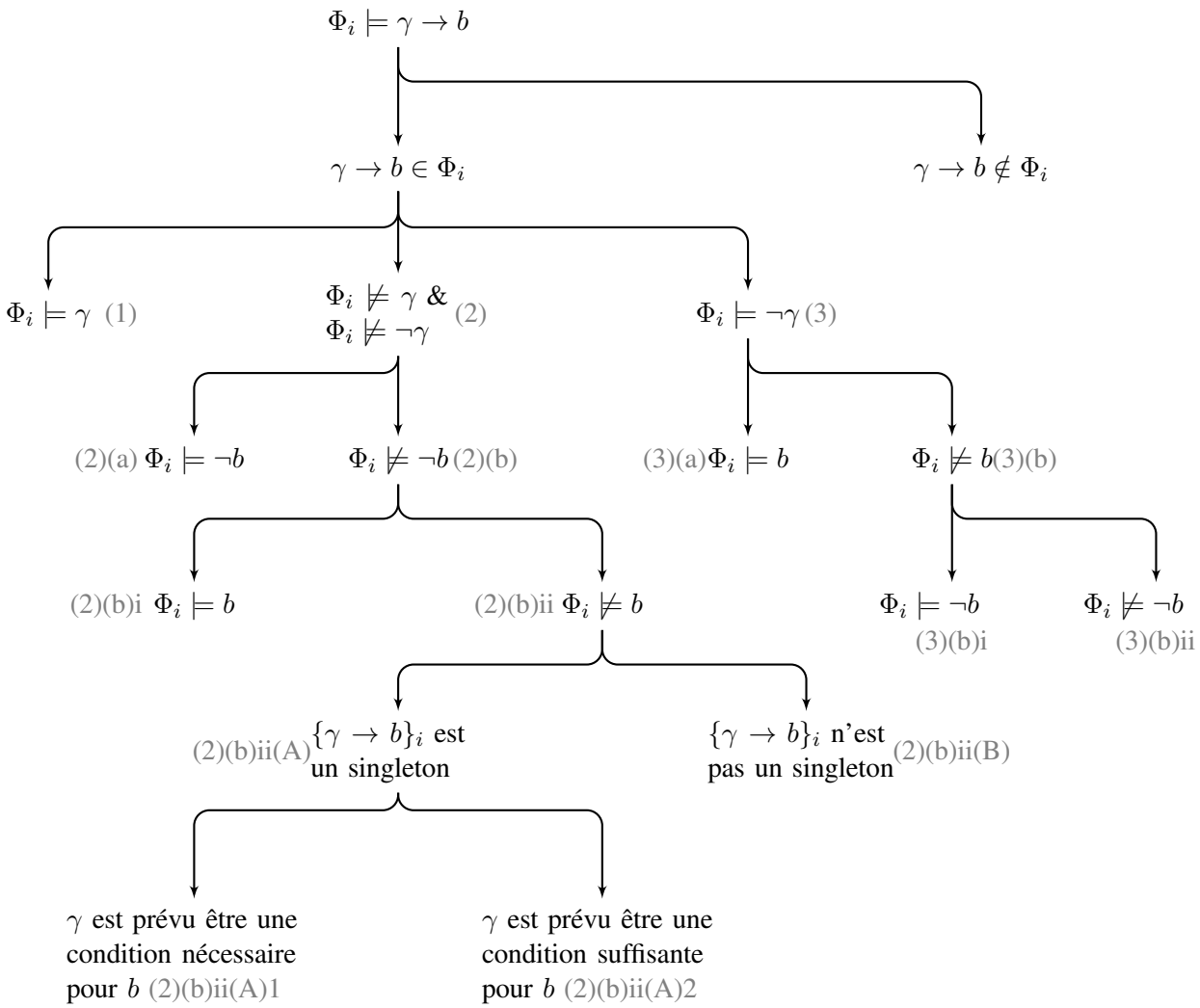

FIGURE 7.1 – Arbre de décision de l'analyse.

2. Quand $\Phi_i \not\models \gamma$ et en même temps $\Phi_i \not\models \neg\gamma$.
   (a) Si $\Phi_i \models \neg b$ alors, par contraposition à la règle $\gamma \to b \in \Phi_i$, on a $\Phi_i \models \neg\gamma$, ce qui contredit l'hypothèse courante ; le cas (2)(a) n'existe alors pas.
   (b) Si $\Phi_i \not\models \neg b$ alors certaines situations peuvent aboutir à des consensus pouvant être problématiques du point de vue de l'agent qui a exprimé Φ_i. Analysons cela :
      i. Si $\Phi_i \models b$ alors l'agent correspondant avait exprimé que b est vrai alors que Φ_i n'implique pas que γ l'est aussi. Il pourrait être possible qu'il rejette tout consensus Γ tel que $\Phi_i \cup \Gamma \models b \wedge \neg\gamma$ parce qu'il aurait oublié de préciser dans Φ_i que γ doit être valide. Considérons l'exemple précédent, supposons que $\Phi_1 = \{$ *coupable* $\to$ *mérite-prison*, *mérite-prison* $\}$. Un tel agent rejettera tout consensus Γ tel que $\Gamma \cup \Phi_1 \models \neg$ *coupable* $\wedge$ *mérite-prison* $\wedge$ (*coupable* $\to$ *mérite-prison*) si il avait oublié que $coupable$ doit y être ajouté. Cependant, à moins qu'il s'agit de cas très spécifiques, il ne semble pas raisonnable de donner à ces consensus un niveau de préférence inférieur. En effet, pour mettre en place ce paradigme dans le cas général, il faudrait supposer, quand Φ_i implique b,

que toute règle dans Φ_i qui a b comme conséquence a aussi son antécédent qui est vrai. En d'autres termes, chaque fois que b est vrai pour un agent, l'antécédent de n'importe quelle règle impliquant b devrait être à vrai afin d'éviter l'existence de consensus qui conduirait à dériver le contraire de cet antécédent. Cette exigence serait très radicale et incompatible avec la représentation de plusieurs règles concernant b ayant des antécédents mutuellement incompatibles. Ainsi, nous ne conservons pas ce cas spécifique comme une raison de déclasser un consensus (y compris la situation où $\gamma \rightarrow b$ est la seule règle ayant b comme conséquence dans Φ_i). Cette politique peut être aussi exprimée comme suit : lorsqu'un agent Φ_i n'exprime pas un raisonnement contrefactuel par une règle dans le sens où il ne s'assure pas que l'antécédent est *faux*, bien qu'il reconnaisse que la conséquence soit *vrai*, nous ne supposons pas ici qu'il rejettera tout consensus selon lequel cette règle exprimerait un raisonnement contrefactuel parce que l'antécédent s'avère être *faux* lorsque le consensus est pris en compte. Cependant, les résultats de cette analyse peuvent être facilement adaptés si cette condition susceptible de déclasser de nombreux consensus possibles doit être adoptée.

ii. Si $\Phi_i \not\models b$ alors plusieurs situations doivent encore être distinguées.

(2)(b)ii(A) D'abord, supposons que Φ_i ne contient aucune autre règle ayant b en conséquence. Dans l'exemple présenté en section 7.1, la situation la plus simple est $\Phi_i = \{$*coupable* $\rightarrow$ *mérite-prison*$\}$. Deux cas différents se produisent selon que l'agent qui a exprimé Φ_i suppose de manière implicite que γ (c'est-à-dire, *coupable* dans l'exemple) est une condition nécessaire ou non pour que b soit dérivable. Nous allons devoir raisonner par cas.

(2)**(b)ii**$(A)(1)$ Supposons que l'agent considère implicitement que γ est une condition nécessaire pour que b soit dérivable. Alors, il pourrait ne pas approuver un consensus Γ tel que $\Phi_i \cup \Gamma \models \neg\gamma \wedge b$. Notons que l'on peut s'attendre à ce qu'il approuve un consensus qui implique b et qui n'implique pas γ mais il est simplement satisfiable avec γ : en effet, un tel consensus offre la possibilité que γ soit vrai et l'agent peut supposer que γ reste une condition implicite, c'est-à-dire, une condition qui n'est pas impliquée par le consensus, pour que b soit vrai.

(2)**(b)ii**$(A)(2)$ Sinon, il pourrait accepter un consensus Γ tel que $\Phi_i \cup \Gamma \models b \wedge \neg\gamma$ uniquement quand il considère que sa règle $\gamma \rightarrow b$ est censée exprimer que γ est une condition suffisante (vs. condition nécessaire) pour dériver b : d'après son point de vue implicite, b devrait être dérivable quand γ est vrai mais d'autres conditions différentes pourraient également suffire à garantir que b soit vrai.

En fait, les informations nécessaires pour démontrer si un consensus peut ou non être interprété comme problématique, c'est-à-dire, quel cas parmi (2)(b)ii$(A)(1)$ and (2)(b)ii$(A)(2)$ s'applique vraiment, peut faire défaut car cela dépend de certaines hypothèses non exprimées par l'agent concerné. Ainsi, la politique que nous suivons consiste à éviter le calcul de consensus tel que $\Phi_i \cup \Gamma \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$ dans les cas (2)(b)ii(A). Lorsqu'il n'y a pas d'autre consensus (maximaux) ou lorsque les autres consensus calculés n'apportent pas de satisfaction au groupe d'agents, le système peut alors calculer un consensus susceptible d'être problématique pour certains agents.

(2)(b)ii(B) Considérons maintenant le cas où Φ_i contient explicitement plusieurs règles différentes de la forme $\gamma_j \rightarrow b$. Rappeler que $\{\gamma_j \rightarrow b\}_i$ désigne l'ensemble des règles dans Φ_i ayant b comme conséquence. Tout d'abord, précisons que nous ne prenons pas en compte les erreurs possibles d'encodage produites lorsque $(\gamma_1 \wedge \gamma_2) \rightarrow b$ est exprimée par deux règles $\gamma_1 \rightarrow b$ et $\gamma_2 \rightarrow b$. Ainsi, chaque γ_j de $\{\gamma_j \rightarrow b\}_i$ traduit une condition suffisante pour impliquer b à partir de Φ_i. Encore une fois, il se pourrait

que Φ_i n'approuve pas un consensus Γ tel que $\Phi_i \cup \Gamma$ implique b et, en même temps, contredit chaque antécédent γ_j de chaque règle $\gamma_j \rightarrow b$ dans Φ_i. Cela se produit lorsque Φ_i considère qu'elle a exprimé toutes les règles possibles pour dériver b et que b ne peut pas être obtenu autrement et que, d'après (2)(b)ii, quand, en même temps, $\Phi_i \not\models b$, $\Phi_i \not\models \neg b$ et il y a une règle dans $\{\gamma_j \rightarrow b\}_i$ telle que $\Phi_i \not\models \gamma_j$ et $\Phi_i \not\models \neg\gamma_j$. Encore une fois, en raison du fait qu'il n'y pas moyen de savoir si l'agent qui a exprimé Φ_i a cette idée en tête, tout consensus Γ tel que $\Gamma \cup \Phi_i$ implique b et, en même temps, $\bigwedge_j \neg\gamma_j$ ne devrait pas être calculé en premier dans ces circonstances. Ces consensus qui pourraient être problématiques pour Φ_i, peuvent alors être calculés quand aucun autre consensus (maximal) n'a été trouvé pour le groupe d'agents.

Clairement l'ensemble des conditions $\Phi_i \not\models b$, $\Phi_i \not\models \neg b$ et l'existence des règles dans $\{\gamma_j \rightarrow b\}_i$ telles que $\Phi_i \not\models \gamma_j$ et $\Phi_i \not\models \neg\gamma_j$ subsume le cas précédent (2)(b)ii(A) où l'ensemble $\{\gamma_j \rightarrow b\}_i$ est un singleton (l'ensemble contient une seule règle). De plus l'ensemble des consensus Γ éventuellement problématiques, tels que $\Phi_i \cup \Gamma \models b \wedge \bigwedge_j \neg\gamma_j$ inclut l'ensemble des consensus potentiellement problématiques dans le cas (2)(b)ii(A) où $\{\gamma_j \rightarrow b\}_i$ est un singleton.

3. Quand $\Phi_i \models \neg\gamma$, Φ_i comporte un raisonnement contrefactuel puisque Φ_i contient également la règle $\gamma \rightarrow b$. Notons aussi que $\nexists\, \Gamma$ tel que $\Gamma \models \gamma$ puisque $\Phi_i \cup \Gamma$ doit être satisfaisable. Nous allons raisonner une fois de plus par cas.
   (a) Si $\Phi_i \models b$ alors il n'y a pas de problème pour l'agent qui a exprimé Φ_i à approuver un consensus Γ tel que $\Phi_i \cup \Gamma \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$ puisque $b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$ est déjà impliqué par Φ_i.
   (b) Si $\Phi_i \not\models b$ alors
      i. Si $\Phi_i \models \neg b$ alors $\nexists\Gamma$ tel que $\Phi_i \cup \Gamma \models b$ et donc il n'y a aucun consensus Γ qui doit être au niveau de préférence inférieur.
      ii. Si $\Phi_i \not\models \neg b$ alors tout consensus Γ tel que $\Phi_i \cup \Gamma \models b \wedge \neg\gamma \wedge (\gamma \rightarrow b)$ pourrait ne pas être approuvé par cet agent lorsqu'il suppose que γ est une condition nécessaire pour que b soit valide.

         Encore une fois, supposons dans une premier temps que $\gamma \rightarrow b$ est une unique règle dans Φ_i qui a b comme conséquence. Du moment que nous n'avons aucune indication sur le fait que l'agent ait ou non l'intention d'exprimer une telle condition comme nécessaire, ces consensus sont possiblement problématiques et ils ne seront pas calculés et sélectionnés en premier lieu. Reconsidérons l'exemple de la section 7.1, quand $\Phi_1 = \{$ *coupable* $\rightarrow$ *mérite-prison*, $\neg$*coupable* $\}$, nous ne sélectionnerons pas en premier un consensus Γ tel que $\Gamma \cup \Phi_1 \models$ *mérite-prison* puisque nous aurions aussi $\Gamma \cup \Phi_1 \models \neg$*coupable* $\wedge$ *mérite-prison* car $\Phi_1 \models \neg$*coupable*.

         Supposons maintenant qu'il y a d'autres règles ayant b comme conséquence dans Φ_i. Contrairement au cas 2(b)ii(B), l'agent implique qu'une seule condition γ_j qui n'est pas satisfaite, car $\Phi_i \models \neg\gamma_j$. En effet, dans le cas 2(b)ii(B), nous avions seulement $\Phi_i \not\models \gamma_j$ mais pas $\Phi_i \models \neg\gamma_j$. Cependant, nous considérons que c'est justifiable d'adopter une position similaire. En effet, si Φ_i approuve un consensus Γ tel que $\Phi_i \cup \Gamma \models b \wedge \bigwedge_j \neg\gamma_j$ alors l'agent accepterait qu'il existe d'autres conditions (indépendantes) qu'il n'a pas exprimées et qui permettent d'avoir b. Encore une fois, nous donnons une préférence moindre à ces consensus, car nous n'avons aucune information sur le fait que l'agent accepterait ou non cela et dès lors ces consensus incluraient, du point de vue de l'agent, un motif de raisonnement contrefactuel qu'il n'a pas envisagé quand il construisait Φ_i.

Ainsi, nous accordons un niveau de préférence inférieur uniquement pour les consensus où nous avons les deux conditions (1) $\Phi_i \not\models b$ et (2) $\Phi_i \not\models \neg b$ réunies ensemble et lorsque (3) on a soit $\Phi_i \not\models \neg\gamma_k$ ou bien $\Phi_i \models \neg\gamma_k$.

Comme $\Phi_i \not\models b$ implique $\Phi_i \not\models \gamma_k$, la condition (3) est inutile dans le sens où elle couvre tous les sous-cas possibles. Par conséquent, les seules situations où certains consensus doivent être relégués, c'est-à-dire, (2)(b)ii et (3)(b)ii, ne doivent pas être différenciés. De plus, les sous-cas, qui découlent de (2)(b)ii et (3)(b)ii qui se produisent lorsque il y a une seule règle avec b comme conséquence dans Φ_i, sont subsumées par les sous-cas il s'agit de plusieurs règles dans Φ_i ayant la conséquence b.

Enfin, discutons du cas où $\Phi_i \models \gamma \rightarrow b$ mais $(\gamma \rightarrow b) \notin \Phi_i$. Puisque $\Delta \models \neg\alpha$ nous avons aussi $\Delta \models \alpha \rightarrow b$ pour tout b, nous ne considérons pas cette situation et nous nous limitons à notre approche basée sur la syntaxe où il y a distinction entre les règles et les formules disjonctives de Φ_i. Toutefois, une façon possible de se débarrasser de ces formules subsumées consisterait à gérer l'ensemble des impliquants premiers de Φ_i au lieu de Φ_i, car ce dernier n'inclut pas de règle subsumée, cependant une telle analyse dépasse le cadre de cette étude.

L'analyse ci-dessus justifie la définition 7.1 pour le concept de *consensus admissible*.

Notons que l'unique facteur syntaxique du cadre logique considéré dans cette analyse est l'existences de règles ayants en conséquence b dans Φ_i. D'un point de vue sémantique, il est facile de voir que l'analyse ci-dessus considère tous les ensembles possibles des interprétations partielles de Φ_i qui assignent des valeurs de vérité à γ et b, et les répertorie dans une ensemble de cas pour lesquels certains consensus devraient recevoir un niveau de préférence inférieur.

7.4 Consensus $max_{\#}$ admissibles et consensus admissibles $max_{\#}$

Naturellement, nous nous intéressons à la notion de maximalité dans les consensus admissibles. Dans ce contexte, maximalité et admissiblité peuvent être combinées de différentes façon. Ici, nous fixons simplement une priorité entre les deux critères. Dans la suite, nous considérons le critère de maximalité par rapport à la cardinalité. L'objectif donc porte sur la recherche d'un consensus maximal qui est admissible, noté consensus $max_{\#}$ admissible, et la recherche d'un consensus admissible qui est maximale, noté consensus admissible $max_{\#}$, parmi tous les consensus admissibles. Formellement, ils sont définis de la manière suivante.

Définition 7.2. Consensus $max_{\#}$ admissible
Un **consensus $max_{\#}$ admissible** Γ pour $\mathcal{S}$ est un consensus $max_{\#}$ pour $\mathcal{S}$ qui est admissible.

Définition 7.3. Consensus admissible $max_{\#}$
Un **consensus admissible $max_{\#}$** Γ pour $\mathcal{S}$ est un consensus admissible pour $\mathcal{S}$ tel que, pour tout consensus admissible Θ pour $\mathcal{S}$, $\#\Gamma \geq \#\Theta$.

Clairement, tout consensus $max_{\#}$ admissible est un consensus admissible $max_{\#}$ tandis que l'inverse n'est pas toujours vrai. Bien que l'existence de consensus admissibles, et ainsi d'un consensus admissible $max_{\#}$, soit garantie, l'existence d'un consensus $max_{\#}$ admissible n'est pas garantie dans le cas général, comme l'illustre l'exemple suivant.

Exemple 7.1. Soit $\mathcal{S} = [\Phi_1, \Phi_2]$ où $\Phi_1 = \{$*coupable* $\rightarrow$ *mérite-prison*$\}$ et $\Phi_2 = \{\neg$*coupable, mérite-prison*$\}$. Comme $\Phi_1 \cup \Phi_2$ est satisfaisable, $\mathcal{S}$ possède un unique consensus $max_{\#}$, c'est-à-dire $\Phi_1 \cup \Phi_2$, qui n'est pas un consensus admissible pour $\mathcal{S}$. Donc, il n'y a pas de consensus $max_{\#}$ admissible pour $\mathcal{S}$. En revanche, il y a deux consensus admissibles $max_{\#}$ pour $\mathcal{S}$: $\{$*coupable* $\rightarrow$ *mérite-prison*, $\neg$*coupable*$\}$ et $\{$*coupable* $\rightarrow$ *mérite-prison*, *mérite-prison*$\}$.

Dans la prochaine section, nous exposons deux différentes techniques : la première pour calculer un consensus $max_{\#}$ admissible et la deuxième pour calculer un consensus admissible $max_{\#}$.

7.5 Calcul d'un consensus $max_{\#}$ admissible

D'abord, focalisons nous sur le calcul d'un consensus $max_{\#}$ admissible. Notons que cela couvre aussi le recherche d'un consensus $max_{\subseteq}$ admissible puisque l'ensemble des consensus $max_{\#}$ est inclus dans l'ensemble des consensus $max_{\subseteq}$. La méthode que nous proposons pour construire un consensus $max_{\#}$ admissible découle de l'approche proposée par (Grégoire *et al.* 2016c) pour le calcul d'un consensus $max_{\#}$. L'idée est simple, elle consiste à itérer la recherche d'un consensus $max_{\#}$, en utilisant l'approche par transformation (Grégoire *et al.* 2016c) jusqu'à en trouver un premier consensus $max_{\#}$ qui satisfait toutes les contraintes de la définition de consensus admissibles, ou sinon jusqu'à ce que tous les consensus $max_{\#}$ soient consultés (examinés). À chaque étape, le consensus $max_{\#}$ est testé pour décider si oui ou non il est admissible. Dans le cas négatif, la procédure bloque le consensus trouvé de sorte qu'il ne sera plus revisité dans les prochaines itérations.

Dans le prochaine section, nous adaptons la procédure de calcul d'un consensus $max_{\#}$ pour fournir, cette fois, un consensus admissible $max_{\#}$.

7.6 Calcul d'un consensus admissible $max_{\#}$

Avant de décrire la méthode pour la recherche d'un consensus admissible $max_{\#}$, nous définissons les *contraintes d'admissibilités* pour Φ. Étant donné que la condition $\Gamma \cup \Phi_i \not\models b \wedge \bigwedge_j \neg\gamma_j$ dans la définition 7.1 est équivalente à ce que $\Gamma \cup \Phi_i$ soit satisfaisable avec $\neg b \vee \bigvee_j \gamma_j$, nous définissons, dans cette section, Φ'_i comme étant l'ensemble des clauses équivalentes à tout $\neg b \vee \bigvee_j \gamma_j$ telles que les conditions supplémentaires de la définition 7.1 soient respectées.

Définition 7.4. (Contraintes d'admissibilité)
Les **contraintes d'admissibilité** de Φ_i, notées Φ'_i, est l'ensemble des clauses de la CNF de conjonction des formules $\neg b \wedge \bigvee_j \gamma_j$ pour tout ensemble non vide $\{\gamma_j \rightarrow b\}_i$ où $\Phi_i \not\models b$ et $\Phi_i \not\models \neg b$.

À partir de la définition 7.1, nous obtenons de manière direct la propriété suivante qui nous sera utile pour adapter la procédure de (Grégoire *et al.* 2016c) au calcul d'un consensus admissible $max_{\#}$.

Propriété 7.3. Soit Γ un consensus pour $\mathcal{S} = [\Phi_1, \ldots, \Phi_n]$. Γ est un consensus admissible de $\mathcal{S}$ si et seulement si $\Gamma \cup \Phi'_i$ est satisfaisable pour tout Φ'_i.

Cette propriété implique que pour calculer un consensus admissible $max_{\#}$ Γ, il suffit de remplacer le test de satisfaisabilité avec Φ_i dans la procédure de (Grégoire *et al.* 2016c) (Algorithme 3.1) par un test de satisfaisabilité avec $\Phi_i \cup \Phi'_i$. Soulignons que Γ à retourner par la procédure est bien un sous-ensemble des clauses dans $\mathcal{S}$ (vs. $[\Phi_1 \cup \Phi'_1, \ldots, \Phi_n \cup \Phi'_n]$). La propriété justifie donc directement un pré-traitement des différents Φ_i pour calculer un consensus admissible $max_{\#}$. Ce pré-traitement traduit dans Φ'_i les situations où des hypothèses implicites possibles de l'agent correspondant pourraient être importantes dans la recherche de consensus admissibles.

Ainsi, notre procédure de calcul d'un consensus admissible $max_{\#}$ hérite de l'efficacité en pratique de le méthode introduite par (Grégoire *et al.* 2016c) pour le calcul d'un consensus $max_{\#}$. Il est intéressant de voir que dans (Grégoire *et al.* 2018c), un autre problème différent de celui étudié ici, est

techniquement parlant, en coïncidence avec le problème de calcul d'un consensus admissible $max_{\#}$. Le problème soulevé dans (Grégoire *et al.* 2018c) est à propos d'agents prudents qui doivent faire face à plusieurs hypothèses qui peuvent être contradictoires. Comme ces agents sont prudents, ils veulent prendre des décisions qui sont compatibles avec chacune de ces hypothèses. Par conséquent, ils doivent fonder leurs propres raisonnements sur des sous-ensembles maximaux de leurs propres informations, qui sont satisfaisables avec chacune de ces hypothèses. De plus, ils doivent également respecter certaines contraintes supplémentaires qui ne sont pas nécessairement prises en compte dans ces sous-ensembles. Techniquement, ce problème consiste à extraire un sous-ensemble $max_{\#}$ (maximal par rapport à la cardinalité) d'un ensemble de formules $\bigcup_{i=1}^{n} \Phi_i$ potentiellement insatisfaisables tel que Γ est satisfaisable avec chaque Φ_i considéré individuellement. Chaque Φ_i est considéré avec un ensemble, notons le Φ'_i, de contraintes que doit satisfaire Γ, sauf que ces contraintes ne sont pas insérées dans Γ. Des expérimentations ont été réalisées dans (Grégoire *et al.* 2018c) sur un large panel de benchmarks de type *timetabling* issus de la compétition internationale PATAT,[16] en utilisant une extension de l'outil RCL (Grégoire et Lagniez 2016b) développé pour le calcul de consensus maximaux. Les résultats obtenus révèlent l'efficacité pratique et le passage à l'échelle de cette approche. Un point intéressant à noter est que le calcul d'un consensus admissible $max_{\#}$ pour $\mathcal{S}$ est techniquement similaire au problème étudié dans (Grégoire *et al.* 2018c), en supposant que les Φ'_i sont définis en suivant la définition 7.4 et calculés lors d'une étape de pré-traitement. Par conséquent, les résultats expérimentaux de (Grégoire *et al.* 2018c) s'appliquent au calcul d'un consensus admissible $max_{\#}$. Ils montrent ainsi la viabilité et l'efficacité pratique de notre méthode de calcul d'un consensus admissible $max_{\#}$.

7.7 Conclusion

Dans cette contribution, nous avons introduit le concept consensus admissible, une extension de la définition de consensus présentée dans (Grégoire *et al.* 2016c). Celui-ci qui évite certaines formes de raisonnement contrefactuel potentiellement indésirables. Cette étude peut être interprétée comme portant sur la définition et la mise en œuvre d'une forme originale de préférence entre les consensus. Cette préférence met en lumière des consensus qui contournent certains éventuels mauvais encodages logiques des bases de connaissance. Nous avons montré que les consensus maximaux admissibles peuvent être calculés grâce à une extension de l'approche introduite par (Grégoire *et al.* 2016c) pour le calcul des consensus maximaux. Comme perspective à ces travaux, nous prévoyons d'intégrer la notion d'admissibilité dans les différents critères de préférences sur les consensus proposés par (Grégoire *et al.* 2016c) afin de permettre le calcul de consensus admissibles maximaux.

16. www.patatconference.org

Conclusions et perspectives

Cette thèse apporte différentes contributions sur le traitement de l'incohérence dans un cadre de logique propositionnelle.

À présent, dressons le résumé de nos résultats ainsi que quelques perspectives qui nous semblent les plus prometteuses. Notons que les travaux de cette thèse ont fait l'objet de publications et communications nationales et internationales (voir (Grégoire *et al.* 2016a), (Grégoire *et al.* 2016b), (Grégoire *et al.* 2017), (Grégoire *et al.* 2018b) et (Grégoire *et al.* 2018a)).

Résumé des contributions

Dans un ordre chronologique, nous avons d'abord travaillé sur l'extraction d'un ensemble maximal d'informations qui soit cohérent avec des contextes mutuellement contradictoires. Nous avons proposé une approche incrémentale originale pour l'extraction d'un tel ensemble (AC-MSS) dans le cadre clausal booléen. Nous avons établi une propriété intéressante qui permet à notre méthode incrémentale d'éviter le parcours systématique de l'espace de recherche qu'effectue l'algorithme incrémental basique et ainsi d'économiser des appels potentiellement coûteux à l'oracle SAT. Les expérimentations que nous avons menées montrent l'efficacité en pratique de notre méthode et prouvent que celle-ci surpasse les techniques de l'état de l'art.

Nos travaux se sont ensuite tournés vers la question de calcul de consensus dans un groupe d'agents négociateurs ou de manière plus générale des sources d'informations possiblement mutuellement conflictuelles. Le concept de consensus est défini par une approche basée sur le cadre propositionnel et peut être caractérisé par différents critères de préférences, comme par exemple le critère de maximalité par rapport à la cardinalité ou l'inclusion ensembliste. Dans cette approche, le consensus est un sous-ensemble des ensembles d'informations qui est cohérent avec chaque source d'informations prise individuellement. En analysant cette propriété, nous avons constaté qu'une correspondance existe entre le concept de consensus et celui d'un AC-MSS. De ce fait, nous avons adopté l'algorithme incrémental pour le calcul d'un AC-MSS pour la recherche d'un consensus maximal par rapport à l'inclusion ensembliste. Ce dernier se montre expérimentalement plus performant, en temps d'execution et espace mémoire, que la méthode générique proposée initialement pour calculer des consensus. En outre, nous avons également introduit et étudié la notion de consensus *admissible* qui est une définition plus raffinée du concept de consensus proposé par (Grégoire *et al.* 2016c). Ainsi, cette contribution porte sur la définition d'une forme originale de préférence entre les consensus filtrant certains d'entre eux qui peuvent ne pas être approuvés par certains agents, en raison d'une représentation logique ad-hoc des connaissances et de paradoxes liés à l'utilisation naturelle mais maladroite de l'implication matérielle.

Nos investigations se sont finalement portées sur le problème d'énumération exhaustive des ensembles minimaux rectificatifs (MCSes) d'une instance CNF inconsistante lorsque leur nombre reste traitable. En effet, l'énumération complète des MCSes/MSSes est une étape utile et parfois primordiale pour mettre en œuvre certaines formes de raisonnement sceptique. Dans le cas d'un raisonneur avec une attitude idéalement sceptique les informations à prendre doivent être dans l'intersection de tous les MSSes. Dans cette contribution, nous avons introduit une technique qui améliore les performances des meilleures

approches pour l'énumération des MCSes d'une formule CNF inconsistante. Cette méthode implémente le paradigme de rotation de modèle et s'appuie sur des propriétés que nous avons trouvées et montrées, permettant ainsi à la méthode de calcul récursif des ensembles de MCSes de manière heuristique et efficace. En pratique, notre algorithme d'énumération de MCSes se montre extrêmement compétitif et se classe parmi les meilleures approches connues à ce jour dans la littérature.

De fait, nous prévoyons de nombreuses perspectives et poursuites possibles de ces travaux.

Perspectives

Les résultats prometteurs que nous avons obtenus dans le calcul d'un consensus maximal par rapport à l'inclusion nous laissent envisager la possibilité d'étendre notre méthode incrémentale pour calculer un consensus maximal par rapport à la cardinalité, également les consensus respectant différents critères de préférences entre agents et leurs clauses. Une autre initiative intéressante serait de pouvoir énumérer tous ou partiellement les consensus maximaux afin d'offrir une plus grande possibilité de choix au groupe d'agents sur le consensus à adopter, et cela grâce à une méthode incrémentale. L'enjeu est alors de trouver des moyens efficaces permettant de réutiliser les calculs précédents pour l'extraction du consensus maximal suivant.

Une autre voie de recherche qui nous semble intéressante est d'exporter l'approche de consensus de (Grégoire *et al.* 2016c) sur laquelle on s'est basé ici, vers un cadre argumentatif. (Caminada et Pigozzi 2011) considèrent une notion de consensus entre les positions des agents exprimées par un étiquetage, étant donné une argumentation abstraite commune. Puisque l'argumentation abstraite peut être encodée en logique propositionnelle (Besnard *et al.* 2014), il serait intéressant de vérifier si cette approche de consensus et son aspect calculatoire pourraient ouvrir de nouvelles perspectives pour de telles investigations sur l'argumentation abstraite. Une autre piste que nous souhaitons explorer consiste à étendre le concept de consensus aux logiques dédiées à la représentation et au raisonnement sur les actions et les plans, comme les logiques temporelles modales ou les fragments booléens du langage PDDL.

L'énumération des ensembles maximaux satisfaisables avec une série de contextes hypothétiques (AC-MSSes) est l'une de nos perspectives à l'issue de cette thèse. Pour ce faire, nous pensons qu'il est tout à fait réalisable d'adopter la technique d'énumération des ensemble minimaux rectificatifs (MCSes) présentée dans le chapitre 4 pour l'énumération des AC-MSSes.

En ce qui concerne l'approche d'énumération des MCSes d'une formule CNF que nous avons proposée, nous pensons que celle-ci peut encore être améliorée. En effet, il pourrait être intéressant d'étendre le processus de rotation de modèle en utilisant des formes de recherche locale pour détecter des clauses de transition supplémentaires. Une étude que nous voulons aussi mener dans le cadre de l'énumération des MCSes concerne la décomposition des instances CNF en composantes connexes afin de calculer localement les MCSes. En effet, des travaux réalisés par (Biere et Sinz 2006) montrent que la résolution du problème SAT peut tirer bénéfice de la décomposition des formules CNF en composantes connexes. Par conséquent, il nous paraît potentiellement avantageux de construire des MCSes localement sur les composantes connexes d'une formule CNF étant donné que les appels au solveur SAT peuvent être moins coûteux que si on faisait appel au solveur SAT sur toute la CNF. Enfin, notons qu'un problème d'énumération qui a longtemps été étudié est celui des cliques maximales (Gély *et al.* 2009). Ainsi, il serait tout à fait intéressant de voir si les algorithmes élaborés dans ce contexte peuvent être réutilisés dans le problème d'énumération des MCSes. Une étude dans cette direction est également envisagée.

Bibliographie

Roberto Javier Asín ACHÁ et Robert NIEUWENHUIS : Curriculum-based course timetabling with SAT and maxsat. *Annals OR*, 218(1):71–91, 2014.

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

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAIS : On freezing and reactivating learnt clauses. *In SAT*, volume 6695 de *Lecture Notes in Computer Science*, pages 188–200. Springer, 2011.

Gilles AUDEMARD, Jean-Marie LAGNIEZ et Laurent SIMON : Improving glucose for incremental SAT solving with assumptions : Application to MUS extraction. *In SAT*, volume 7962 de *Lecture Notes in Computer Science*, pages 309–317. Springer, 2013.

Gilles AUDEMARD et Laurent SIMON : Predicting learnt clauses quality in modern SAT solvers. *In IJCAI*, pages 399–404, 2009.

Gilles AUDEMARD et Laurent SIMON : Lazy clause exchange policy for parallel SAT solvers. *In SAT*, volume 8561 de *Lecture Notes in Computer Science*, pages 197–205. Springer, 2014.

Fahiem BACCHUS, Jessica DAVIES, Maria TSIMPOUKELLI et George KATSIRELOS : Relaxation search : A simple way of managing optional clauses. *In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 835–841, 2014.

Fahiem BACCHUS et George KATSIRELOS : Using minimal correction sets to more efficiently compute minimal unsatisfiable sets. *In Proc. of CAV 2015*, pages 70–86, 2015.

Fahiem BACCHUS et George KATSIRELOS : Finding a collection of muses incrementally. *In Proc. of CPAIOR 2016*, pages 35–44, 2016.

James BAILEY et Peter J. STUCKEY : Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. *In Practical Aspects of Declarative Languages, 7th International Symposium, PADL 2005, Long Beach, CA, USA, January 10-11, 2005, Proceedings*, pages 174–186, 2005.

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

Paul BEAME, Henry A. KAUTZ et Ashish SABHARWAL : Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

Anton BELOV, Inês LYNCE et João MARQUES-SILVA : Towards efficient MUS extraction. *AI Commun.*, 25(2):97–116, 2012.

Anton BELOV et João MARQUES-SILVA : Accelerating MUS extraction with recursive model rotation. *In Proc. of FMCAD 2011*, pages 37–40, 2011.

Salem BENFERHAT, Jonathan BEN-NAIM, Robert JEANSOULIN, Mahat KHELFALLAH, Sylvain LAGRUE, Odile PAPINI, Nic WILSON et Eric WÜRBEL : Belief revision of GIS systems : The results of rev !gis. *In Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU 2005, Barcelona, Spain, July 6-8, 2005, Proceedings*, pages 452–464, 2005.

Philippe BESNARD, Sylvie DOUTRE et Andreas HERZIG : Encoding argument graphs in logic. *In Information Processing and Management of Uncertainty in Knowledge-Based Systems - 15th International Conference, IPMU 2014, Montpellier, France, July 15-19, 2014, Proceedings, Part II*, pages 345–354, 2014.

Philippe BESNARD, Éric GRÉGOIRE et Jean-Marie LAGNIEZ : On computing maximal subsets of clauses that must be satisfiable with possibly mutually-contradictory assumptive contexts. *In AAAI*, pages 3710–3716. AAAI Press, 2015.

Philippe BESNARD et Anthony HUNTER : *Elements of Argumentation*. MIT Press, 2008.

Armin BIERE et Carsten SINZ : Decomposing SAT problems into connected components. *JSAT*, 2 (1-4):201–208, 2006.

Elazar BIRNBAUM et Eliezer L. LOZINSKII : Consistent subsets of inconsistent systems : structure and behaviour. *J. Exp. Theor. Artif. Intell.*, 15(1):25–46, 2003.

Elizabeth BLACK, Sanjay MODGIL et Nir OREN, éditeurs. *Theory and Applications of Formal Argumentation - Second International Workshop, TAFA 2013, Beijing, China, August 3-5, 2013, Revised Selected papers*, volume 8306 de *Lecture Notes in Computer Science*, 2014. Springer.

Daniel G. BOBROW : Special issue non non-monotonic logics. *Artificial Intelligence*, 13(1-2), 1980.

Andrei BONDARENKO, Phan Minh DUNG, Robert A. KOWALSKI et Francesca TONI : An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

George BOOLE : *Les lois de la pensée*. Mathesis, 1854.

Abdelhamid BOUDANE, Saïd JABBOUR, Lakhdar SAIS et Yakoub SALHI : A sat-based approach for mining association rules. *In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2472–2478, 2016.

Aaron R. BRADLEY : IC3 and beyond : Incremental, inductive verification. *In Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, page 4, 2012.

Renato BRUNI : Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete Appl. Math.*, 130(2):85–100, août 2003. ISSN 0166-218X.

Renato BRUNI : On exact selection of minimally unsatisfiable subformulae. *Ann. Math. Artif. Intell.*, 43 (1):35–50, 2005.

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

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

Martin CAMINADA et Gabriella PIGOZZI : On judgment aggregation in abstract argumentation. *Autonomous Agents and Multi-Agent Systems*, 22(1):64–102, 2011.

Martin W. A. CAMINADA et Dov M. GABBAY : A logical account of formal argumentation. *Studia Logica*, 93(2-3):109–145, 2009.

Federico CERUTTI, Paul E. DUNNE, Massimiliano GIACOMIN et Mauro VALLATI : Computing preferred extensions in abstract argumentation : a sat-based approach. *CoRR*, abs/1310.4986, 2013.

Günther CHARWAT, Wolfgang DVORÁK, Sarah Alice GAGGL, Johannes Peter WALLNER et Stefan WOLTRAN : Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.

Brian F. CHELLAS : Basic conditional logic. *Journal of Philosophical Logic*, 4(2):133–153, 1975.

Zhi-Zhong CHEN et Seinosuke TODA : The complexity of selecting maximal solutions. *Information and Computation*, 119(2):231–239, 1995.

Edmund M. CLARKE, Armin BIERE, Richard RAIMI et Yunshan ZHU : Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

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

James M. CRAWFORD et Andrew B. BAKER : Experimental results on the application of satisfiability algorithms to scheduling problems. *In Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2.*, pages 1092–1097, 1994.

Ian DAVIDSON, S. S. RAVI et Leonid SHAMIS : A sat-based framework for efficient constrained clustering. *In Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 94–105, 2010.

Jessica DAVIES et Fahiem BACCHUS : Solving MAXSAT by solving a sequence of simpler SAT instances. *In Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, pages 225–239, 2011.

Jessica DAVIES et Fahiem BACCHUS : Exploiting the power of mip solvers in maxsat. *In Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, pages 166–181, 2013a.

Jessica DAVIES et Fahiem BACCHUS : Postponing optimization to speed up MAXSAT solving. *In Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 247–262, 2013b.

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

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

Johan de KLEER : An assumption-based TMS. *Artif. Intell.*, 28(2):127–162, 1986.

Johan de KLEER, Alan K. MACKWORTH et Raymond REITER : Characterizing diagnoses and systems. *Artif. Intell.*, 56(2-3):197–222, 1992.

Johan de KLEER et Brian C. WILLIAMS : Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

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.

Phan Minh DUNG : On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

Phan Minh DUNG, Paolo MANCARELLA et Francesca TONI : Computing ideal sceptical argumentation. *Artif. Intell.*, 171(10-15):642–674, 2007.

Wolfgang DVORÁK, Matti JÄRVISALO, Johannes Peter WALLNER et Stefan WOLTRAN : Cegartix : A sat-based argumentation system. *Pragmatics of SAT Workshop*, 2012.

Doroty EDGINGTON : Conditionals. *In* E.N. Zalta (ED.), éditeur : *The Stanford Encyclopedia of Philosophy*. 2008.

Niklas EÉN et Niklas SÖRENSSON : An extensible sat-solver. *In Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003a.

Niklas EÉN et Niklas SÖRENSSON : Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003b.

Eithan EPHRATI et Jeffrey S. ROSENSCHEIN : Deriving consensus in multiagent systems. *Artif. Intell.*, 87(1-2):21–74, 1996.

Alexander FELFERNIG, Monika SCHUBERT et Christoph ZEHENTNER : An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, 26(1):53–62, 2012.

Eduardo L. FERMÉ et Sven Ove HANSSON : AGM 25 years - twenty-five years of research in belief change. *J. Philosophical Logic*, 40(2):295–331, 2011.

Zhaohui FU et Sharad MALIK : On solving the partial MAX-SAT problem. *In SAT*, volume 4121 de *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

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

M. R. GAREY et David S. JOHNSON : *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Olivier GAUWIN, Sébastien KONIECZNY et Pierre MARQUIS : Conciliation through iterated belief merging. *J. Log. Comput.*, 17(5):909–937, 2007.

Alain GÉLY, Lhouari NOURINE et Bachir SADI : Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447–1459, 2009.

Matthew L. GINSBERG, éditeur. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-45-1.

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

Carla P. GOMES, Bart SELMAN et Nuno CRATO : Heavy-tailed distributions in combinatorial search. *In Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*, pages 121–135, 1997.

Carla P. GOMES, Bart SELMAN et Henry A. KAUTZ : Boosting combinatorial search through randomization. *In Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 431–437, 1998.

Éric GRÉGOIRE : Skeptical inheritance can be more expressive. *In ECAI*, pages 326–332, 1990.

Éric GRÉGOIRE, Yacine IZZA et Jean-Marie LAGNIEZ : Extraction d'un sous-ensemble maximal d'informations qui soit cohérent avec des contextes mutuellement contradictoires. *In Journées Francophones de Programmation par Contraintes, JFPC'16*, 2016a.

Éric GRÉGOIRE, Yacine IZZA et Jean-Marie LAGNIEZ : On the extraction of one maximal information subset that does not conflict with multiple contexts. *In AAAI*, pages 3404–3410. AAAI Press, 2016b.

Éric GRÉGOIRE, Yacine IZZA et Jean-Marie LAGNIEZ : On computing one max_subset inclusion consensus. *In ICTAI*, pages 838–845. IEEE Computer Society, 2017.

Éric GRÉGOIRE, Yacine IZZA et Jean-Marie LAGNIEZ : Boosting mcses enumeration. *In IJCAI*, pages 1309–1315. ijcai.org, 2018a.

Éric GRÉGOIRE, Yacine IZZA et Du ZHANG : On admissible consensuses. *International Journal on Artificial Intelligence Tools*, 27(1):1–20, 2018b.

Éric GRÉGOIRE, Sébastien KONIECZNY et Jean-Marie LAGNIEZ : On consensus extraction. *In IJCAI*, pages 1095–1101. IJCAI/AAAI Press, 2016c.

Éric GRÉGOIRE et Jean-Marie LAGNIEZ : A computational approach to consensus-finding. *In ECAI*, volume 285 de *Frontiers in Artificial Intelligence and Applications*, pages 795–801. IOS Press, 2016a.

Éric GRÉGOIRE et Jean-Marie LAGNIEZ : RCL : an A.I. tool for computing maximal consensuses. *International Journal on Artificial Intelligence Tools*, 25(4):1–10, 2016b.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Bertrand MAZURE : Improving MUC extraction thanks to local search. *CoRR*, abs/1307.3585, 2013.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Bertrand MAZURE : An experimentally efficient method for (mss, comss) partitioning. *In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2666–2673, 2014a.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Bertrand MAZURE : A general artificial intelligence approach for skeptical reasoning. *In Artificial General Intelligence - 7th International Conference, AGI 2014, Quebec City, QC, Canada, August 1-4, 2014. Proceedings*, pages 33–42, 2014b.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Du ZHANG : Consensus-finding that preserves mutually conflicting hypothetical information from a same agent. *AI Commun.*, 31(3):303–317, 2018c.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Extracting muses. *In ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pages 387–391, 2006.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Boosting a complete technique to find MSS and MUS thanks to a local search oracle. *In IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2300–2305, 2007a.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Boosting a complete technique to find MSS and MUS thanks to a local search oracle. *In Proc. of IJCAI 2007*, pages 2300–2305, 2007b.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Local-search extraction of muses. *Constraints*, 12(3):325–344, 2007c.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Using local search to find msses and muses. *European Journal of Operational Research*, 199(3):640–646, 2009.

Walter HAMSCHER, Luca CONSOLE et Johan de KLEER, éditeurs. *Readings in Model-based Diagnosis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992. ISBN 1-55860-249-6.

Fred HEMERY, Christophe LECOUTRE, Lakhdar SAIS et Frédéric BOUSSEMART : Extracting mucs from constraint networks. *In ECAI*, volume 141 de *Frontiers in Artificial Intelligence and Applications*, pages 113–117. IOS Press, 2006.

C. A. R. HOARE : Algorithm 64 : Quicksort. *Commun. ACM*, 4(7):321, 1961.

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

Audun JØSANG : The consensus operator for combining beliefs. *Artif. Intell.*, 141(1/2):157–170, 2002.

Ulrich JUNKER : QUICKXPLAIN : preferred explanations and relaxations for over-constrained problems. *In Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 167–172, 2004.

Antonis C. KAKAS et Pavlos MORAITIS : Argumentation based decision making for autonomous agents. *In The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 883–890, 2003.

Henry A. KAUTZ et Bart SELMAN : Planning as satisfiability. *In ECAI*, pages 359–363, 1992.

Henry A. KAUTZ et Bart SELMAN : Pushing the envelope : Planning, propositional logic and stochastic search. *In Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2.*, pages 1194–1201, 1996.

Philip KILBY, John K. SLANEY, Sylvie THIÉBAUX et Toby WALSH : Backbones and backdoors in satisfiability. *In AAAI*, pages 1368–1373. AAAI Press / The MIT Press, 2005.

Sébastien KONIECZNY et Éric GRÉGOIRE : Logic-based approaches to information fusion. *Information Fusion*, 7(1):2–3, 2006.

Oliver KULLMANN : Constraint satisfaction problems in clausal form II : minimal unsatisfiability and conflict structure. *Fundam. Inform.*, 109(1):83–119, 2011.

Jean-Marie LAGNIEZ, Emmanuel LONCA et Jean-Guy MAILLY : Coquiaas : Applications de la programmation par contraintes 'a l'argumentation abstraite. *Actes JFPC*, 2015a.

Jean-Marie LAGNIEZ, Emmanuel LONCA et Jean-Guy MAILLY : Coquiaas : A constraint-based quick abstract argumentation solver. *In Proc. of ICTAI 2015*, pages 928–935, 2015b.

João LEITE, Tran Cao SON, Paolo TORRONI, Leon van der TORRE et Stefan WOLTRAN, éditeurs. *Computational Logic in Multi-Agent Systems - 14th International Workshop, CLIMA XIV, Corunna, Spain, September 16-18, 2013. Proceedings*, volume 8143 de *Lecture Notes in Computer Science*, 2013. Springer.

Paolo LIBERATORE : Redundancy in logic I : CNF propositional formulae. *Artif. Intell.*, 163(2):203–232, 2005.

Mark H. LIFFITON, Alessandro PREVITI, Ammar MALIK et Joao MARQUES-SILVA : Fast, flexible MUS enumeration. *Constraints*, 21(2):223–250, 2016.

Mark H. LIFFITON et Karem A. SAKALLAH : Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.

Mark H. LIFFITON et Karem A. SAKALLAH : Generalizing core-guided max-sat. *In SAT*, volume 5584 de *Lecture Notes in Computer Science*, pages 481–494. Springer, 2009.

Eliezer L. LOZINSKII : Resolving contradictions : A plausible semantics for inconsistent systems. *J. Autom. Reasoning*, 12(1):1–32, 1994.

Inês LYNCE et João MARQUES-SILVA : SAT in bioinformatics : Making the case with haplotype inference. *In Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, pages 136–141, 2006.

David MAKINSON et Karl SCHLECHTA : Floating conclusions and zombie paths : Two deep difficulties in the directly skeptical approach to defeasible inheritance nets. *Artif. Intell.*, 48(2):199–209, 1991.

João MARQUES-SILVA : Computing minimally unsatisfiable subformulas : State of the art and future directions. *Multiple-Valued Logic and Soft Computing*, 19(1-3):163–183, 2012.

João MARQUES-SILVA, Federico HERAS, Mikolás JANOTA, Alessandro PREVITI et Anton BELOV : On computing minimal correction subsets. *In IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

Joao MARQUES-SILVA et Mikolás JANOTA : On the query complexity of selecting few minimal sets. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:31, 2014.

João MARQUES-SILVA, Mikolás JANOTA, Alexey IGNATIEV et António MORGADO : Efficient model based diagnosis with maximum satisfiability. *In IJCAI*, pages 1966–1972. AAAI Press, 2015.

John MCCARTHY : Circumscription - A form of non-monotonic reasoning. *Artif. Intell.*, 13(1-2):27–39, 1980.

John MCCARTHY : Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.*, 28(1):89–116, 1986.

Carlos MENCÍA, Alexey IGNATIEV, Alessandro PREVITI et Joao MARQUES-SILVA : MCS extraction with sublinear oracle queries. *In SAT*, volume 9710 de *Lecture Notes in Computer Science*, pages 342–360. Springer, 2016.

Carlos MENCÍA, Alessandro PREVITI et João MARQUES-SILVA : Literal-based MCS extraction. *In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1973–1979, 2015.

Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : engineering an efficient sat solver. *In DAC '01 : Proceedings of the 38th annual Design Automation Conference*, pages 530–535, New York, NY, USA, 2001a. ACM. ISBN 1-58113-297-2.

Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : Engineering an efficient SAT solver. *In Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001b.

Alexander NADEL : Boosting minimal unsatisfiable core extraction. *In Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 221–229, 2010.

Alexander NADEL, Vadim RYVCHIN et Ofer STRICHMAN : Efficient MUS extraction with resolution. *In Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 197–200, 2013.

Alexander NADEL, Vadim RYVCHIN et Ofer STRICHMAN : Accelerated deletion-based extraction of minimal unsatisfiable cores. *JSAT*, 9:27–51, 2014.

Nina NARODYTSKA, Nikolaj BJØRNER, Maria-Cristina MARINESCU et Mooly SAGIV : Core-guided minimal correction set and core enumeration. *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1353–1361. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

Alexander NÖHRER, Armin BIERE et Alexander EGYED : Managing SAT inconsistencies with HUMUS. *In VaMoS*, pages 83–91. ACM, 2012.

Chanseok OH : Between SAT and UNSAT : the fundamental difference in CDCL SAT. *In Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 9340 de *Lecture Notes in Computer Science*, pages 307–323. Springer, 2015.

Yoonna OH, Maher N. MNEIMNEH, Zaher S. ANDRAUS, Karem A. SAKALLAH et Igor L. MARKOV : AMUSE : a minimally-unsatisfiable subformula extractor. *In Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 518–523, 2004.

Christos H. PAPADIMITRIOU : *Computational complexity*. Addison-Wesley, 1994a.

Christos H. PAPADIMITRIOU : *Computational complexity*. Addison-Wesley, 1994b. ISBN 978-0-201-53082-7.

Luís Moniz PEREIRA, Carlos Viegas DAMÁSIO et José Júlio ALFERES : Debugging by diagnosing assumptions. *In AADEBUG*, volume 749 de *Lecture Notes in Computer Science*, pages 58–74. Springer, 1993.

Sylvain PERIFEL : *Complexité Algorithmique*. Ellipses, 2014.

Alessandro PREVITI et João MARQUES-SILVA : Partial MUS enumeration. *In Proc. of AAAI 2013*, 2013.

Alessandro PREVITI, Carlos MENCÍA, Matti JÄRVISALO et Joao MARQUES-SILVA : Improving MCS enumeration via caching. *In SAT*, volume 10491 de *Lecture Notes in Computer Science*, pages 184–194. Springer, 2017.

Alessandro PREVITI, Carlos MENCÍA, Matti JÄRVISALO et Joao MARQUES-SILVA : Premise set caching for enumerating minimal correction subsets. *In AAAI*. AAAI Press, 2018.

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

Raymond REITER : A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

Raymond REITER : A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, 1987.

Wei REN, R. W. BEARD et E. M. ATKINS : A survey of consensus problems in multi-agent coordination. *Proceedings of the 2005, American Control Conference, 2005.*, pages 1859–1864 vol. 3, 2005.

John Alan ROBINSON : A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

Emanuele Di ROSA, Enrico GIUNCHIGLIA et Marco MARATEA : Solving satisfiability problems with preferences. *Constraints*, 15(4):485–515, 2010.

Francesca ROSSI, Peter van BEEK et Toby WALSH, éditeurs. *Handbook of Constraint Programming*, volume 2 de *Foundations of Artificial Intelligence*. Elsevier, 2006.

Vadim RYVCHIN et Ofer STRICHMAN : Faster extraction of high-level minimal unsatisfiable cores. *In Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, pages 174–187, 2011.

Nicolas SCHWIND et Pierre MARQUIS : On consensus in belief merging. *In AAAI*. AAAI Press, 2018.

Bart SELMAN, Hector J. LEVESQUE et David MITCHELL : Gsat : A new method for solving hard satisfiability problems. *In Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.

Claude Elwood SHANNON : A symbolic analysis of relay and switching circuits. Thesis (m.s.), Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1940.

João P. Marques SILVA et Inês LYNCE : On improving MUS extraction algorithms. *In Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, pages 159–173, 2011.

João P. Marques SILVA et Karem A. SAKALLAH : GRASP - a new search algorithm for satisfiability. *In ICCAD*, pages 220–227, 1996.

João P. Marques SILVA et Karem A. SAKALLAH : Boolean satisfiability in electronic design automation. *In Proceedings of the 37th Conference on Design Automation, Los Angeles, CA, USA, June 5-9, 2000.*, pages 675–680, 2000.

Alexander SMITH, Andreas G. VENERIS et Anastasios VIGLAS : Design diagnosis using boolean satisfiability. *In Proceedings of the 2004 Conference on Asia South Pacific Design Automation : Electronic Design and Solution Fair 2004, Yokohama, Japan, January 27-30, 2004*, pages 218–223, 2004.

Paul R. STEPHAN, Robert K. BRAYTON et Alberto L. SANGIOVANNI-VINCENTELLI : Combinational test generation using satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(9): 1167–1176, 1996.

Ofer STRICHMAN : Pruning techniques for the sat-based bounded model checking problem. *In Correct Hardware Design and Verification Methods, 11th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2001, Livingston, Scotland, UK, September 4-7, 2001, Proceedings*, pages 58–70, 2001.

G. TSEITIN : On the complexity of proofs in propositional logics. 2, 1983.

G.S. TSEITIN : On the complexity of derivations in the propositional calculus. *In* H.A.O. SLESENKO, éditeur : *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

Tomás E. URIBE et Mark E. STICKEL : Ordered binary decision diagrams and the davis-putnam procedure. *In CCL '94 : Proceedings of the First International Conference on Constraints in Computational Logics*, pages 34–49, London, UK, 1994. Springer-Verlag. ISBN 3-540-58403-X.

Miroslav N. VELEV et Randal E. BRYANT : Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *In Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 226–231, 2001.

Jesse WHITTEMORE, Joonyoung KIM et Karem A. SAKALLAH : SATIRE : A new incremental satisfiability engine. *In Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 542–545, 2001.

Eric WÜRBEL, Robert JEANSOULIN et Odile PAPINI : Revision : an application in the framework of GIS. *In KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000.*, pages 505–515, 2000.

T YASUMOTO et T OKUGAWA : Rokk. *SAT Competition*, 2014.

Résumé

Cette thèse étudie une approche possible de l'intelligence artificielle pour la détection et le curage d'informations perverties dans les bases de connaissances des objets et composants intelligents en informatique ubiquitaire. Cette approche est traitée d'un point de vue pratique dans le cadre du formalisme SAT ; il s'agit donc de mettre en œuvre des techniques de filtrage d'incohérences dans des bases contradictoires. Plusieurs contributions sont apportées dans cette thèse. Premièrement, nous avons travaillé sur l'extraction d'un ensemble maximal d'informations qui soit cohérent avec une série de contextes hypothétiques. Nous avons proposé une approche incrémentale pour le calcul d'un tel ensemble (AC-MSS). Deuxièmement, nous nous sommes intéressés à la tâche d'énumération des ensembles maximaux satisfaisables (MSS) ou leurs complémentaires les ensembles minimaux rectificatifs (MCS) d'une instance CNF insatisfaisable. Dans cette contribution, nous avons introduit une technique qui améliore les performances des meilleures approches pour l'énumération des MSS/MCS. Cette méthode implémente le paradigme de rotation de modèle qui permet de calculer des ensembles de MCS de manière heuristique et efficace. Finalement, nous avons étudié une notion de consensus permettant réconcilier des sources d'informations. Cette forme de consensus peut être caractérisée par différents critères de préférence, comme le critère de maximalité. Une approche incrémentale de calcul d'un consensus maximal par rapport à l'inclusion ensembliste a été proposée. Nous avons également introduit et étudié la concept de consensus admissible qui raffine la définition initialement proposée du concept de consensus.

Mots-clés: Informatique ubiquitaire, intelligence artificielle, SAT, MSS/MCS, AC-MSS, consensus

Abstract

This thesis studies a possible approach of artificial intelligence for detecting and filtering inconsistent information in knowledge bases of intelligent objects and components in ubiquitous computing. This approach is addressed from a practical point of view in the SAT framework ;it is about implementing a techniques of filtering inconsistencies in contradictory bases. Several contributions are made in this thesis. Firstly, we have worked on the extraction of one maximal information set that must be satisfiable with multiple assumptive contexts. We have proposed an incremental approach for computing such a set (AC-MSS). Secondly, we were interested about the enumeration of maximal satisfiable sets (MSS) or their complementary minimal correction sets (MCS) of an unsatisfiable CNF instance. In this contribution, a technique is introduced that boosts the currently most efficient practical approaches to enumerate MCS. It implements a model rotation paradigm that allows the set of MCS to be computed in an heuristically efficient way. Finally, we have studied a notion of consensus to reconcile several sources of information. This form of consensus can obey various preference criteria, including maximality one. We have then developed an incremental algorithm for computing one maximal consensus with respect to set-theoretical inclusion. We have also introduced and studied the concept of admissible consensus that refines the initial concept of consensus.

Keywords: Ubiquitous computing, artificial intelligence, SAT, MSS/MCS, AC-MSS, consensus