Aix-Marseille Université

École Doctorale de Mathématiques et d'Informatique de Marseille

Faculté des Sciences

Laboratoire des Sciences de l'Information et des Systèmes

Thèse présentée pour obtenir le grade universitaire de docteur en informatique

André Abramé

Max-résolution et apprentissage pour la résolution du problème de satisfiabilité maximum

Sous réserve de l'avis des rapporteurs,
soutenance prévue le 25 Septembre 2015 devant le jury :

Jin-Kao Hao Université d'Angers	Rapporteur
Chu Min Li Université de Picardie Jules Verne	Rapporteur
Felip Manyà Artificial Intelligence Research Institute (IIIA, CSIC), Bellaterra, Spain	Examinateur
Philippe Jégou Aix-Marseille Université	Directeur de thèse
Djamal Habet Aix-Marseille Université	Directeur de thèse

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Liste des notations

Symboles

$\Phi|_I$ la formule Φ simplifiée par l'interpréation I 18

$\overline{x_i}$ négation de la variable x_i 15

$c_j|_I$ la clause c_j simplifiée par l'interpréation I 17

$l_i|_I$ le littéral l_i simplifié par l'interpréation I 17

B

$break(x_i)$ nombre de nouvelles clauses falsifiées si on flippe la variable x_i 30

C

$cout(\Phi, I)$ somme des poids des clauses de Φ falsifiées par l'interprétation I . . . 37

E

$eval(\Phi, I)$ fonction d'évaluation donnant le nombre de clauses de Φ falsifiées par l'interprétation complète I 30

F

$flip(I, x_i)$ interprétation résultante du flip de la variable x_i dans l'interprétation complète I 29

M

$make(x_i)$ nombre de nouvelles clauses satisfaites si on flippe la variable x_i . . . 30

N

$\mathcal{N}(I)$ ensemble des interprétations voisines de l'interprétation complète I 29

$neigh_G(l_i)$ l'ensemble des voisins du littéral l_i dans le graphe d'implications G . 20

O

$optimum(\Phi)$ somme minimale des poids des clauses falsifiées pour la formule Φ 37

P

$poids(c_j)$ poids associé à la clause c_j dans une formule Max-SAT valué 36

$pred_G(l_i)$ l'ensemble des prédecesseurs du littéral l_i dans le graphe d'implications G 20

R

S

V

Introduction générale

Au cours du XX^{e} siècle, plusieurs facteurs ont entraîné un bouleversement en profondeur des sociétés humaines. La population mondiale a quadruplé, le développement des moyens de communication et de transport a entraîné une augmentation et une globalisation des échanges, tandis que l'émergence de l'électronique puis de l'informatique a conduit à l'utilisation d'objets de plus en plus élaborés à toutes les échelles de nos sociétés. Dans ce contexte, les défis à relever sont de plus en plus nombreux et complexes, allant de la rationalisation de l'utilisation des ressources naturelles à la gestion et l'organisation des réseaux de communication, en passant par l'optimisation du transport de marchandises ou la validation et la vérification des programmes informatiques. L'automatisation du traitement et de la résolution de ces problèmes est donc un élément important du développement et du bon fonctionnement de nos sociétés.

Une alternative à la création de méthodes de résolution *ad hoc* pour chacun de ces problèmes consiste à élaborer des formalismes globaux permettant de les représenter et des méthodes de résolution générales dépendantes uniquement de ces formalismes et non de la nature du problème initial. Ce champs d'étude, la *programmation par contraintes*, est à l'intersection de plusieurs disciplines telles que l'intelligence artificielle, la logique, la théorie de la complexité ou l'optimisation combinatoire. Il regroupe des formalismes permettant de représenter un ensemble de choix – les *variables* – et les relations entre ces choix – les *contraintes* –. À chacun de ces formalismes est associé un ensemble de problèmes, tels la *décision* (déterminer une interprétation des variables qui ne viole aucune contrainte), l'*optimisation* (déterminer une interprétation des variables qui minimise le nombre de contraintes violées), etc. Des méthodes de résolution sont développées pour chacun de ces problèmes, avec pour objectif de permettre la résolution d'instances toujours plus grandes en un temps toujours plus court.

Parmi les problèmes de la programmation par contraintes, ceux basés sur la logique propositionnelle [Boo54] ont été particulièrement étudiés. Le problème de satisfiabilité propositionnelle (SAT) a reçu une attention importante, attention motivée par sa position centrale en théorie de la complexité (c'est le premier problème à avoir été démontré comme étant NP-complet [Coo71, Lev73]) et par le nombre important de ses applications, aussi bien issues de problèmes réels qu'académiques. Il consiste à déterminer si une formule propositionnelle en forme normale conjonctive en entrée (c'est-à-dire une conjonction de clauses, chaque clause étant une disjonction de littéraux et chaque littéral étant une variable propositionnelle ou sa négation) est valide, autrement dit s'il existe une interprétation des variables satisfaisant toutes les clauses de la formule. De nombreux algorithmes ont été mis au point pour permettre la résolution en un temps raisonnable de formules de très grande taille. Ces algorithmes utilisent des règles d'inférence puissantes pour simplifier la formule initiale ou l'enrichir de nouvelles informations. Depuis le début

des années 2000, l'utilisation intensive de techniques d'apprentissage dans les solveurs de type *Conflict Driven Clause Learning* (CDCL) s'est avérée particulièrement efficace sur les instances fortement structurées, comme le sont généralement celles issues de la transformation de problèmes réels.

Dans cette thèse, nous nous intéressons au problème Max-SAT (*maximum satisfiability*) qui peut être vu comme une version optimisation du problème SAT. Il consiste à trouver, pour une formule propositionnelle en forme normale conjonctive en entrée, une interprétation des variables de la formule qui maximise le nombre de clauses satisfaites. Il a lui aussi de nombreuses applications réelles dans des domaines variés comme le routage [JKS95, XRS02, FM06], la bio-informatique [SBS05, GL12], la synthèse de circuit [GP95], le diagnostic [DG12], etc. Les variantes Max-SAT valué et Max-SAT partiel, qui permettent respectivement d'établir des priorités entre les contraintes et d'empêcher la violation de certaines contraintes, ont un pouvoir de représentation plus grand que le problème Max-SAT classique. Parmi les méthodes de résolution complètes pour Max-SAT, c'est-à-dire celles qui garantissent l'optimalité des solutions qu'elles renvoient, on peut distinguer trois approches. La première, basée sur des algorithmes de type séparation et évaluation (*branch and bound*) [HLO07, HLO08, LMP07, LMMP10, Küg10], est particulièrement efficace sur les instances aléatoires et *crafted* non-partielles. La seconde est basée sur des appels itératifs à un solveur SAT [FM06, HMS11, KZFH12, MML14, MML12, MML13, MDM14, NB14] et elle excelle sur les instances issues de problèmes industriels. La dernière est basée sur la reformulation des instances en programmation linéaire en nombres entiers (*integer linear programming*, ILP) [Ach09, AG13] et domine les autres approches sur les instances *crafted* partielles. Les approches incomplètes quant à elles ne peuvent prouver l'optimalité des solutions qu'elles renvoient et sont majoritairement basées sur des algorithmes de recherche locale [AH12, CLTS14, HJ90, SHS03, YI98].

Les solveurs de type séparation et évaluation (e.g. WMAXSATZ [LMP07, LMMP10], AKMAXSAT [Küg10], MINIMAXSAT [HLO07, HLO08]) parcourent l'espace des interprétations en construisant un arbre de recherche. À chaque nœud de cet arbre, ils comparent la meilleure solution trouvée jusqu'à présent (la borne supérieure, UB) à une sous-estimation de la meilleure solution accessible dans la branche courante de l'arbre (la borne inférieure, LB). Si $LB \geq UB$, alors il n'est pas utile d'explorer la branche actuelle de l'arbre de recherche et ils réalisent un retour-arrière. L'efficacité de ces solveurs repose sur leur capacité à "élaguer" l'arbre de recherche, c'est-à-dire à ignorer le plus tôt possible les branches ne pouvant contenir de solution meilleure que la borne supérieure.

Le calcul de la borne inférieure est un des éléments primordiaux des solveurs de type séparation et évaluation. Il est réalisé à chaque nœud de l'arbre de recherche et donc il représente une part importante du temps de résolution. Sa qualité détermine le nombre de nœuds explorés dans l'arbre de recherche. Il consiste à réaliser une sous-estimation du nombre de sous-ensembles inconsistants disjoints présents dans la formule simplifiée par l'interprétation courante. Les solveurs de type séparation et évaluation récents détectent les sous-ensembles inconsistants par des méthodes basées sur la propagation unitaire, puis ils les transforment pour s'assurer qu'ils ne soient comptés qu'une fois. Une des méthodes de transformation utilisées est basée sur la règle de la *max-résolution*. Elle préserve l'équivalence de la formule et permet une forme d'apprentissage lorsque les modifications qu'elle apporte sont conservées. Elle a cependant des inconvénients, puisqu'elle peut entraîner une augmentation rapide de la taille de la formule. Pour cette raison, elle n'est

appliquée que sur en moyenne 20% des sous-ensembles inconsistants et les modifications ne sont conservées que dans la sous-partie de l'arbre de recherche.

Bien qu'ils dominent les autres types de solveurs sur les instances faiblement structurées, les solveurs de type séparation et évaluation sont perfectibles sur plusieurs aspects. Les traitements qu'ils appliquent lors du calcul de la borne inférieure sont très redondants, puisque les mêmes sous-ensembles inconsistants peuvent être redétectés et retransformés à de nombreux nœuds de l'arbre de recherche. De plus, ils sont notoirement inefficaces sur les instances structurées telles que celles issues de problèmes réels.

Dans ce contexte, l'apprentissage paraît une piste de recherche intéressante puisqu'il permettrait à la fois de réduire la redondance dans le calcul de la borne inférieure et de mieux prendre en compte la structure des instances en exploitant les informations accumulées au fil de la résolution. Nous avons donc étudié le fonctionnement des deux composants principaux utilisés dans le calcul de la borne inférieure et permettant de réaliser un apprentissage : la propagation unitaire et la max-résolution. Les objectifs visés sont multiples :

- améliorer le calcul de la borne inférieure, soit en réduisant le temps de traitement qui y est consacré, soit en améliorant la précision de l'estimation,
- contrôler le principal inconvénient connu des transformations par max-résolution : l'augmentation de la taille de la formule
- mieux comprendre l'impact de ces transformations sur le fonctionnement des solveurs de type séparation et évaluation
- à plus long terme, augmenter l'apprentissage

Les contributions réalisées durant cette thèse vont dans le sens de ces objectifs. Nous avons proposé un nouveau schéma d'application de la propagation unitaire [AH14d, AH15b] qui prend en considération toutes les sources de propagation des variables, réduisant ainsi la redondance lors de la détection des sous-ensembles inconsistants. Nous avons montré que l'ordre d'application des étapes de max-résolution lors de la transformation des sous-ensembles inconsistants a un impact sur le nombre et la taille des clauses de compensation ajoutées à la formule. En se basant sur cette observation, nous avons introduit une nouvelle heuristique [AH14a, AH15a] permettant de limiter l'augmentation de la taille de la formule inhérente à l'application des transformations par max-résolution. Nous avons introduit une nouvelle méthode de transformation des sous-ensembles inconsistants détectés lors du calcul de la borne inférieure, en utilisant la max-résolution de manière locale à chaque nœud de l'arbre de recherche [AH14c, AH14b]. Nous avons également proposé une extension du schéma d'apprentissage [AH14e, AH15d] avant d'étudier l'impact des transformations par max-résolution sur l'efficacité de la propagation unitaire [AH15c]. Nous avons implémenté un nouveau solveur séparation et évaluation, AHMAXSAT, pour évaluer empiriquement l'impact de nos contributions. Lors de la *Max-SAT Evaluation 2014* qui compare les performances des solveurs, AHMAXSAT a été classé premier dans trois des neufs catégories d'instances et second dans une quatrième. Enfin, nous avons également montré que le mécanisme de détection et de transformation des sous-ensembles inconsistants par propagation unitaire et max-résolution pouvait être transposé dans un algorithme de recherche locale pour simplifier la formule et ainsi améliorer son efficacité [AH12, AH13].

Ce document est divisé en deux parties. La première, composée des chapitres 1 et 2, est consacrée à la présentation de l'étude bibliographique que nous avons réalisé. Nous

présentons nos contributions dans la seconde partie. Les chapitres 3 à 7 introduisent les contributions liées aux solveurs séparation et évaluation et au calcul de la borne inférieure. Nous donnons une description globale de notre solveur AHMAXSAT dans le chapitre 8 et nous le comparons empiriquement à certains des meilleurs solveurs complets pour Max-SAT. Le chapitre 9 présente l'intégration de la propagation unitaire et de la max-résolution dans un algorithme de recherche locale.

Première partie

État de l'art

Cette première partie est consacrée à la présentation de l'étude bibliographique que nous avons réalisé. Nous décrivons dans le chapitre 1 le contexte dans lequel se situe l'étude du problème Max-SAT. Nous y faisons une brève introduction à la théorie de la complexité. Nous rappelons la notion de complexité d'un algorithme et présentons les classes de complexité auxquelles appartiennent les problèmes SAT et Max-SAT. Puis, nous donnons les notions de logique propositionnelle nécessaires à la définition des objets considérés dans les problèmes SAT et Max-SAT : les formules propositionnelles en forme normale conjonctive. Nous définissons formellement le problème SAT et rappelons des éléments du vocabulaire qui lui est associé. Enfin, nous présentons les principales règles d'inférence utilisées dans le cadre de la résolution de SAT et détaillons deux classes de méthodes de résolution : les solveurs DPLL et la recherche locale.

Le chapitre 2 est consacré au problème Max-SAT lui-même. Nous y rappelons sa définition et introduisons ses variantes Max-SAT valué et Max-SAT partiel. Nous discutons de l'applicabilité des règles d'inférence pour SAT à Max-SAT et introduisons les règles propres à Max-SAT. Nous détaillons le fonctionnement des solveurs de type séparation et évaluation et en particulier les méthodes utilisées pour calculer la borne inférieure. Nous discutons également de l'application des algorithmes de recherche locale à Max-SAT avant de passer en revue brièvement les autres approches existantes pour résoudre Max-SAT. Enfin, nous discutons de l'évaluation empirique des solveurs Max-SAT et introduisons les benchmarks et la méthodologie expérimentale qui sera utilisée dans la seconde partie de ce document.

Chapitre 1

Préliminaires

Nous présentons dans ce chapitre les principales notions de base nécessaires à la compréhension de cette thèse. Nous faisons tout d'abord une brève introduction à la théorie de la complexité avec pour objectif de permettre au lecteur de saisir le contexte entourant les problèmes SAT et Max-SAT et, par là même, les enjeux liés à leur résolution. La

seconde partie est dédiée à la présentation du problème de satisfiabilité propositionnelle SAT. Nous y décrivons la syntaxe et la sémantique des formules propositionnelles avant de définir formellement le problème SAT. Puis, nous présentons les principales règles d'inférence qui peuvent lui être appliquées avant de décrire deux types de méthodes de résolution : les algorithmes de type DPLL et la recherche locale. L'objectif visé ici n'est pas de faire un état de l'art exhaustif des méthodes de résolution pour SAT, mais plutôt de présenter certaines techniques et méthodes de résolution transposables (directement ou indirectement) au problème Max-SAT.

1.1 Théorie de la complexité

Depuis la création des premières machines à calculer à la fin du XIXème siècle, une réflexion théorique s'est engagée sur la capacité de ces machines à résoudre les problèmes qui leurs sont soumis. L'efficacité des méthodes de résolution est un élément central de cette réflexion. C'est dans ce cadre que s'inscrit l'étude de la théorie de la complexité. Elle permet d'établir une corrélation entre la taille des données en entrée et la quantité de ressources nécessaires à un algorithme pour les traiter et, de manière plus globale, elle vise à classifier les problèmes selon leur difficulté.

Nous faisons dans cette section une brève introduction à la théorie de la complexité, avec pour objectif de permettre au lecteur de comprendre comment évaluer l'efficacité d'un algorithme et la difficulté d'un problème. Ces notions permettront de mieux saisir les enjeux liés à la résolution des problèmes SAT et Max-SAT. Pour une présentation plus complète de la théorie de la complexité, nous invitons le lecteur à se référer aux ouvrages de Garey et Johnson [GJ79], Papadimitriou [Pap94] ou d'Arora et Barak [AB09].

1.1.1 Complexité d'un algorithme

Un des rôles premiers de la théorie de la complexité est d'évaluer les quantités de ressources nécessaires à un algorithme en fonction de la taille des données (de l'instance) qu'il a à traiter. Les ressources considérées sont généralement le temps d'exécution et l'espace mémoire utilisés, mais on peut aussi inclure d'autres ressources selon le domaine d'application de l'algorithme évalué (par exemple les accès réseaux). On notera $R(A, D)$ la quantité de ressource R nécessaire à l'algorithme A pour résoudre le problème D et $\mathcal{D}_{A,n}$ l'ensemble des données de taille n que peut prendre en entrée l'algorithme A.

Le comportement d'un algorithme peut varier selon les données qu'il a à traiter (indépendamment de la taille de celles-ci). Ainsi, pour deux jeux de données $D, D' \in \mathcal{D}_{A,n}$, on n'a pas nécessairement $R(A, D) = R(A, D')$. Pour quantifier les ressources R nécessaires à l'algorithme A en fonction uniquement de la taille des données en entrée, on peut donc considérer :

— le pire des cas possible, noté $R_p(A, n)$, tel que :

$$R_p(A, n) = max(\{R(A, D) \quad \forall D \in \mathcal{D}_{A,n}\})$$

— le cas moyen, noté $R_m(A, n)$, tel que

$$R_m(A, n) = \frac{\sum_{D \in \mathcal{D}_{A,n}} R(A, D)}{|\mathcal{D}_{A,n}|}$$

On notera que déterminer le cas moyen nécessite une connaissance statistique de la structure des données en entrée. Dans la suite de cette thèse, nous considérerons toujours la complexité dans le pire des cas.

Enfin, on s'intéresse moins à la quantité exacte de ressources nécessaires à l'algorithme qu'à son comportement asymptotique. On utilise donc les notations de Landau pour définir la complexité d'un algorithme A pour une ressources R par une fonction qui domine asymptotiquement $R_p(A, n)$.

Définition 1 (Complexité algorithmique)**.** Soit A un algorithme et $f(n)$ une fonction définie sur un entier positif. On dit que la complexité algorithmique dans le pire des cas de A pour une ressource R est $\mathcal{O}(f(n))$ s'il existe des constantes c et n_0 telles que :

$$\forall n > n_0, \quad R_p(A, n) < c \times f(n)$$

On peut définir de la même manière la complexité dans le cas moyen.

Nous dirons d'un algorithme de complexité $\mathcal{O}(f(n))$ pour une ressource R qu'il est linéaire (resp. polynomial) pour R si $f(n)$ est une fonction linéaire (resp. un polynôme).

Le tableau 1.1 donne l'ordre de grandeur de l'évolution du temps d'exécution d'un algorithme suivant sa complexité et la taille de la donnée en entrée. À titre de comparaison, l'âge estimé de l'univers est de l'ordre de 10^{17} secondes [BNV+13]. On voit que les temps de résolution sont très élevés pour les complexités exponentielle et factorielle, et ce même avec des tailles de données raisonnables.

Complexité	Type de complexité	Temps d'exécution (secondes)		
		$n = 10$	$n = 100$	$n = 1000$
1	constante	1	1	1
$log(n)$	logarithmique	1	2	3
n	linéaire	10	100	1000
n^2	quadratique (polynomiale	100	10^4	10^6
n^3	cubique (polynomiale	1000	10^6	10^9
e^n	exponentielle	10^4	10^{43}	10^{434}
$n!$	factorielle	10^6	10^{157}	10^{2567}

TABLE 1.1 – Ordre de grandeur du temps d'exécution d'un algorithme selon sa complexité et la taille n de la donnée en entrée. On suppose qu'une opération atomique est réalisée en 10^{-1} seconde.

De ces observations découle une question : existe-t-il un algorithme polynomial (ou linéaire) quel que soit le problème qu'on cherche à résoudre ? Et, dans le cas contraire, peut-on malgré tout concevoir un algorithme efficace en pratique ? Nous présentons dans la prochaine section quelques éléments de réponse apportés par la théorie de la complexité.

1.1.2 Complexité des problèmes

En informatique théorique, un *problème* est une question pouvant être soumise à un système de calcul tel que nos ordinateurs. Il prend en entrée un jeu de données (ou *instance*) et fournit en sortie une réponse dont le format dépend du type de problème. On appellera méthode de résolution, ou *solveur*, un algorithme permettant de résoudre un problème donné.

On peut distinguer les problèmes selon le type de questions auxquelles ils cherchent à répondre. Ainsi, un *problème de décision* associe à l'instance en entrée une question à laquelle on peut répondre par *oui* ou *non*. Un *problème d'optimisation* consiste à trouver, parmi un ensemble de possibilités, la meilleure solution selon un objectif prédéfini. D'autres types de problèmes existent, comme les *problèmes d'énumération*, etc.

Nous nous intéressons ici à un des autres champs d'étude de la théorie de la complexité qui consiste à déterminer la complexité des problèmes eux-mêmes, c'est-à-dire leurs difficultés intrinsèques. Dans la suite de cette section, nous introduisons tout d'abord un système de calcul théorique permettant de modéliser les capacités des ordinateurs actuels : la machine de Turing. Puis, nous définissons des classes de complexité qui regroupent des problèmes selon leurs difficultés.

1.1.2.1 Machines de Turing

Une machine de Turing est une machine théorique modélisant les capacités de calcul des ordinateurs actuels. Elle est composée d'une mémoire et d'un module de traitement. La mémoire est un ruban de taille infini, divisé en cases contenant chacune un symbole d'un alphabet fini. Le module de traitement est constitué de :

- une tête de lecture/écriture qui lit, écrit et se déplace sur le ruban.
- un registre d'état qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est fini.
- une table d'actions qui indique, en fonction de l'état courant et du symbole lu sur le ruban, l'état suivant de la machine et le mouvement de la tête de lecture (immobile, vers la droite ou vers la gauche du ruban).

Les principales différences entre les machines de Turing et les ordinateurs actuels concernent la mémoire. On considère que le ruban des machines de Turing est de taille infinie. Ce n'est évidemment pas le cas des mémoires (vives ou disques durs) des ordinateurs. De plus, l'accès à la mémoire est séquentiel dans les machines de Turing (à partir d'une position du ruban, seules les cases de droite et de gauche sont accessibles) tandis qu'il est aléatoire (l'information est accessible directement dans la mémoire à partir de son adresse) dans les ordinateurs actuels.

Malgré ces différences, les machines de Turing sont un modèle universel de machine à calculer. Ainsi, la thèse de Church-Turing postule que tout ce qui est calculable peut l'être avec une machine de Turing (ou tout autre modèle de calcul équivalent). De manière formelle, une machine de Turing peut être définie de la manière suivante.

Définition 2 (Machine de Turing déterministe)**.** Une machine de Turing peut être définie comme un septuplé $(Q, \Gamma, b, \Sigma, q_0, \delta, F)$ avec :

- Q l'ensemble (fini) des états que peut prendre la machine,
- Γ l'alphabet (fini) des symboles qui peuvent être écrits sur la bande,
- $b \in \Gamma$ un symbole particulier, dit "blanc", qui représente l'état initial des cases de la bande,
- Σ l'alphabet des symboles de la donnée en entrée (on a $\Sigma \subseteq (\Gamma \setminus \{b\})$),
- $q_0 \in Q$ l'état initial de la machine,
- δ une application de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ et
- $F \subseteq Q$ l'ensemble des états terminaux de la machine.

Une telle machine de Turing est déterministe : à chaque couple (état, symbole), la fonction δ fait correspondre un unique triplé (état, symbole, déplacement). Nous définissons maintenant les machines de Turing non-déterministes.

Définition 3 (Machine de Turing non-déterministe)**.** Une machine de Turing non-déterministe peut être définie comme un septuplé $(Q, \Gamma, b, \Sigma, q_0, \delta, F)$ avec $Q, \Gamma, b, \Sigma, q_0, F$ définis comme précédemment et δ une relation binaire de $Q \times \Gamma$ dans $Q \times \Gamma \times \{\leftarrow, \rightarrow\}$.

A chaque couple (état, symbole), une machine de Turing non-déterministe fait correspondre un ensemble de triplés (état, symbole, déplacement). Intuitivement, une telle machine est capable de se décupler (de passer d'un fil d'exécution à plusieurs) sans limitation.

Pour les machines de Turing (déterministes ou non-déterministes), on mesure le temps d'exécution (et donc la complexité) des algorithmes en fonction du nombre de transitions qu'ils réalisent.

1.1.2.2 Classes de complexité

Nous utilisons maintenant les machines de Turing déterministe et non-déterministe pour définir des classes de complexité, c'est-à-dire des ensembles de problèmes de difficulté comparable. Puis nous introduisons la notion de réduction polynomial, qui permet de comparer la difficulté de deux problèmes et d'établir des liens entre les classes de complexité.

Définition 4 (Classe P)**.** La classe P (pour *polynomial time*) inclut tous les problèmes de décision qui peuvent être résolus en temps polynomial sur une machine de Turing déterministe.

Définition 5 (Classe NP)**.** La classe NP (pour *non-deterministic polynomial time*) inclut tous les problèmes de décision qui peuvent être résolus en temps polynomial sur une machine de Turing non-déterministe.

Définition 6 (C-complet et C-difficile)**.** Soit C une classe de complexité. Un problème est C-difficile s'il est au moins aussi difficile que tous les problèmes de la classe C. Un problème est dit C-complet s'il est C-difficile et qu'il appartient à la classe C.

Ainsi, la classe NP-complet regroupe les problèmes les plus difficiles de la classe NP tandis que la classe NP-difficile regroupe tous les problèmes qui sont au moins aussi difficiles que ceux de la classe NP.

On compare la difficulté des problèmes en utilisant la notion de réduction polynomiale définie ci-dessous.

Définition 7 (Réduction polynomiale)**.** Soit P_1 et P_2 deux problèmes de décision. Une réduction polynomiale de P_1 à P_2 est une méthode de complexité polynomiale qui permet de transformer toute instance p_1 du problème P_1 en une instance p_2 du problème P_2 telle que si p_1 est vrai (resp. faux) dans P_1, alors p_2 est vrai (resp. faux) dans P_2. On écrira $P_1 \leq_p P_2$.

S'il existe une réduction polynomiale d'un problème P_1 vers un problème P_2, alors on peut en déduire que P_2 est au moins aussi difficile que P_1. Et si $P_1 \leq_p P_2$ et $P_2 \leq_p P_1$,

alors les problèmes P_1 et P_2 sont de difficulté équivalente et ils appartiennent aux mêmes classes de complexité. La réduction polynomiale est une relation d'ordre dans l'ensemble des problèmes algorithmiques. Elle permet d'établir les liens suivants entre les classes de complexité introduites précédemment : P $\subseteq$ NP et NP-complet $\subseteq$ NP-difficile. En revanche, un problème NP-difficile n'est pas nécessairement dans NP.

Un des problèmes ouverts de la théorie de la complexité consiste à établir si P=NP, autrement dit si tout problème de NP peut être résolu en temps polynomial. La figure 1.1 montre les relations entre les classes P, NP, NP-complet et NP-difficile dans les deux cas de figure.

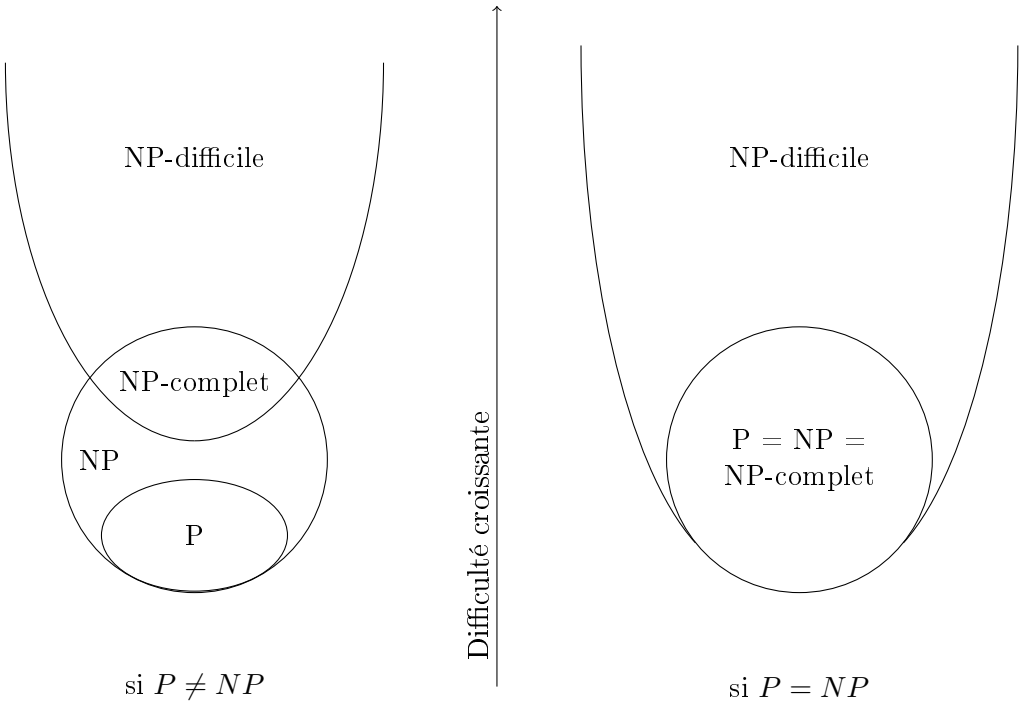


FIGURE 1.1 – Diagramme de Euler des classes P, NP, NP-complet et NP-difficile

1.2 Le problème SAT

L'étude du problème Max-SAT est indissociable de celle du problème de satisfiabilité propositionnelle SAT. L'attention portée à ce dernier a conduit à l'élaboration de nombreuses méthodes de résolution et une partie d'entre elles peuvent être adaptés pour être appliquées dans le cadre du problème Max-SAT. Par exemple, le parcours de l'espace des interprétations réalisé par les solveurs de type séparation et évaluation pour Max-SAT est similaire à celui des solveurs de type DPLL pour SAT. De la même manière, les algorithmes de recherche locale pour SAT peuvent être appliqués à des instances Max-SAT moyennant quelques modifications mineures.

Nous présentons donc dans cette section le problème SAT. Nous rappelons tout d'abord la syntaxe et la sémantique de la logique propositionnelle qui permet de décrire les objets considérés par les problèmes SAT et Max-SAT : les formules en forme

normale conjonctive. Puis nous définissons formellement le problème SAT et rappelons son contexte. Nous passons en revue quelques unes des règles d'inférence qui peuvent lui être appliquées avant de détailler le fonctionnement de deux familles de méthodes de résolution : les algorithmes de type DPLL et la recherche locale.

1.2.1 Logique propositionnelle

Depuis l'Antiquité, l'homme a cherché à formaliser sa pensée en construisant des modèles permettant de représenter les idées ou les connaissances et leurs relations. Ce champs d'étude, la *logique*, est à la jonction de plusieurs disciplines : la philosophie, les mathématiques et plus récemment l'informatique théorique. Parmi les formalismes proposés au cours du temps, la logique propositionnelle [Boo54] occupe une place particulière. Elle est un des premiers formalismes à adopter une structure algébrique et elle constitue encore aujourd'hui la base de nombreux systèmes logiques plus complexes. Nous décrivons ici sa syntaxe et sa sémantique, avant de définir les formules en forme normale conjonctive qui seront considérées dans le cadre des problèmes SAT et Max-SAT.

1.2.1.1 Syntaxe

L'alphabet de la logique propositionnelle est composé de :
— constantes *vrai* (ou 1, V, $\top$) et $faux$ (ou 0, F, $\bot$)
— variables propositionnelles pouvant prendre les valeurs *vrai* ou $faux$
— l'opérateur unaire de négation $\neg$ ou $^-$ (barre au dessus)
— l'opérateur binaire de conjonction $\wedge$
— l'opérateur binaire de disjonction $\vee$
— l'opérateur binaire d'implication logique $\Rightarrow$
— l'opérateur binaire d'équivalence $\Leftrightarrow$
— les symboles de priorité (et)

Cet alphabet permet de construire des formules propositionnelles selon les règles décrites ci-dessous.

Définition 8 (Formule propositionnelle)**.** Soit $X = \{x_1, \ldots, x_n\}$ un ensemble de variables propositionnelles. Une formule propositionnelle est construite récursivement selon les règles suivantes :
— les variables de X, $\top$ et $\bot$ sont des formules,
— si Φ est une formule, alors (Φ) est une formule,
— si Φ est une formule, alors $\neg\Phi$ est une formule,
— si Φ et Φ' sont des formules, alors :
  — $\Phi \wedge \Phi'$ est une formule,
  — $\Phi \vee \Phi'$ est une formule,
  — $\Phi \Rightarrow \Phi'$ est une formule,
  — $\Phi \Leftrightarrow \Phi'$ est une formule,

Les parenthèses déterminent l'ordre d'évaluation des éléments des formules. En leur absence, les opérateurs sont évalués selon l'ordre de priorité suivant : $\Leftrightarrow$, $\Rightarrow$, $\vee$, $\wedge$ et $\neg$.

1.2.1.2 Sémantique

Maintenant que nous avons défini la syntaxe des formules propositionnelles, nous allons nous intéresser à leur sémantique. Interpréter les variables d'une formule propositionnelle consiste à leur donner (affecter) une valeur $vrai$ ou $faux$.

Définition 9 (Interprétation)**.** Une interprétation d'un ensemble de variables propositionnelles X est une application σ de X dans $\{vrai, faux\}$. Elle est dite *complète* si σ fait correspondre à chaque variable de X une valeur et *partielle* sinon.

La notion d'interprétation peut être étendue aux formules. Ainsi, pour une formule Φ définie sur X et σ une interprétation (partielle ou complète) des variables de X, $\sigma(\Phi)$ est la formule obtenue en remplaçant les variables par leur valeur et en appliquant les règles décrites dans la table 1.2.

$\sigma(\Phi)$	$\sigma(\Phi')$	$\sigma(\top)$	$\sigma(\bot)$	$\sigma(\neg\Phi)$	$\sigma(\Phi \wedge \Phi')$	$\sigma(\Phi \vee \Phi')$	$\sigma(\Phi \Rightarrow \Phi')$	$\sigma(\Phi \Leftrightarrow \Phi')$
$vrai$	$vrai$	$vrai$	$faux$	$faux$	$vrai$	$vrai$	$vrai$	$vrai$
$vrai$	$faux$	$vrai$	$faux$	$faux$	$faux$	$vrai$	$faux$	$faux$
$faux$	$vrai$	$vrai$	$faux$	$vrai$	$faux$	$vrai$	$vrai$	$faux$
$faux$	$faux$	$vrai$	$faux$	$vrai$	$faux$	$faux$	$vrai$	$vrai$

TABLE 1.2 – Table de vérité des opérateurs de la logique propositionnelle.

Nous utiliserons le vocabulaire suivant. On dira que la formule $\sigma(\Phi)$ est le résultat de la *simplification* de la formule Φ par l'interprétation σ. Φ est *satisfaite* par σ si $\sigma(\Phi) = vrai$. Nous dirons alors que σ est un *modèle* de Φ. De la même manière, Φ est *falsifiée* par σ si $\sigma(\Phi) = faux$. Une formule propositionnelle Φ' est une *conséquence logique* de Φ (notée $\Phi \models \Phi'$) ssi tout modèle de Φ est aussi un modèle de Φ'. Φ et Φ' sont *équivalentes* ssi $\Phi \models \Phi'$ et $\Phi' \models \Phi$, c'est-à-dire si tout modèle de Φ est un modèle de Φ' et inversement. Nous appellerons *espace de recherche* l'ensemble $\mathcal{I}$ des interprétations possibles des variables de X. On peut noter que $|\mathcal{I}| = 2^{|X|}$.

1.2.1.3 Forme normale conjonctive

Dans le cadre des problèmes SAT et Max-SAT, nous nous intéressons à un format particulier de formules propositionnelles : la forme normale conjonctive. Nous définissons tout d'abord les éléments constituant ces formules avant de donner le vocabulaire et les notations qui seront utilisés dans la suite de cette thèse. Nous considérons dans toute la suite de cette section un ensemble $X = \{x_1, \ldots, x_n\}$ de variables propositionnelles.

Définition 10 (Littéral)**.** Un littéral l est une variable $x_i \in X$ ou sa négation $\overline{x_i}$.

Définition 11 (Clause)**.** Soit $l_1, \ldots, l_k$ des littéraux de variables de X. Une clause c est une disjonction de littéraux $c = l_1 \vee \cdots \vee l_k$. Dans la suite de ce document, nous noterons les clauses sous formes d'ensembles de littéraux $c = \{l_1, \ldots, l_k\}$.

Nous appellerons les clauses contenant respectivement un, deux et trois littéraux *unitaires*, *binaires* et *ternaires*.

Définition 12 (clause tautologique)**.** Une clause est dite tautologique si elle contient un littéral et sa négation.

Propriété 1. Une clause tautologique est satisfaite par toute interprétation complète.

Définition 13 (clause sous-sommée)**.** Soit deux clause c_i et c_j. On dit que c_i sous-somme c_j ssi $c_i \subseteq c_j$.

Définition 14 (Formule en forme normal conjonctive)**.** Une formule Φ en forme normale conjonctive (*conjunctive normal form*, CNF) définie sur l'ensemble de variables X est une conjonction de clauses $\Phi = c_1 \wedge \cdots \wedge c_m$. Dans la suite de ce document, nous noterons les formules CNF sous formes de multi-ensembles de clauses $\Phi = \{c_1, \ldots, c_m\}$.

Remarque 1. Toute formule propositionnelle peut être convertie en une formule CNF équivalente [Tse68].

Sauf indication contraire, nous considérerons dans la suite que toute formule est en forme normale conjonctive. Pour un littéral l, nous noterons $var(l)$ la variable correspondante. Pour une clause c, nous noterons $var(c)$ l'ensemble des variables apparaissant dans la clause c. Formellement, $var(c) = \{var(l_i) \text{ t.q. } l_i \in c\}$. Cette notation peut être étendue à tout ensemble de clauses ϕ (et donc aux formules) comme suit : $var(\phi) = \bigcup_{c_i \in \phi} var(c_i)$.

Nous représenterons les interprétations sous forme d'ensemble de littéraux I ne pouvant contenir un littéral et sa négation. Formellement, pour une interprétation σ :

$$I = \{x_i \text{ t.q. } x_i \in X \text{ et } \sigma(x_i) = vrai\} \cup \{\overline{x_j} \text{ t.q. } x_j \in X \text{ et } \sigma(x_j) = faux\}$$

Nous dirons qu'une interprétation I satisfait un littéral l_i ssi $l_i \in I$ et qu'elle falsifie l_i ssi $\overline{l_i} \in I$. De la même manière, I satisfait une clause c_j ssi elle satisfait au moins l'un de ses littéraux et elle falsifie c_j ssi elle falsifie tous ses littéraux. Les clauses vides, notées $\square$, sont toujours falsifiées. Une formule CNF Φ est satisfaite par une interprétation I ssi I satisfait toutes ses clauses et elle est falsifiée par I ssi I falsifie au moins une de ses clauses. Nous utiliserons les notations suivantes. Pour un littéral l_i, on notera $l_i|_I$ le résultat de la simplification de l_i par l'interprétation I. Formellement :

$$l_i|_I = \begin{cases} vrai & \text{si } l_i \in I \\ faux & \text{si } \overline{l_i} \in I \\ l_i & \text{sinon} \end{cases}$$

De la même manière, nous noterons $c_j|_I$ le résultat de l'application d'une interprétation I sur une clause c_j. Formellement :

$$c_j|_I = \begin{cases} vrai & \text{si } \exists l_i \in c_j,\ l_i \in I \\ faux \text{ (ou } \square) & \text{si } \forall l_i \in c_j,\ \overline{l_i} \in I \\ \{l_i \in c_j \text{ t.q. } \{l_i, \overline{l_i}\} \cap I = \emptyset\} & \text{sinon} \end{cases}$$

Nous dirons qu'un littéral l_i satisfait par I (i.e. $l_i \in I$) *satisfait* c_j si $l_i \in c_j$ et qu'il *réduit* c_j si $\overline{l_i} \in c_j$. Enfin, nous noterons $\Phi|_I$ la formule obtenue en appliquant une interprétation unitaire $I = \{l_i\}$ sur une formule Φ. Formellement :

$$\Phi|_I \ = \ \{c_j \text{ t.q. } c_j \in \Phi \text{ et } \{l_i, \overline{l_i}\} \cap c_j = \emptyset\} \ \cup \ \{c_j \setminus \{\overline{l_i}\} \text{ t.q. } c_j \in \Phi \text{ et } \overline{l_i} \in c_j\}$$

Cette notation peut être étendue à tout interprétation $I = \{l_1, l_2, \ldots, l_k\}$:

$$\Phi|_I \ = \ (\ldots((\Phi|_{\{l_1\}})|_{\{l_2\}})\ldots|_{\{l_k\}})$$

Lorsqu'une formule n'admet pas de modèle, il est souvent utile de pouvoir localiser plus précisément la ou les causes de l'inconsistance, en identifiant un sous-ensemble des clauses de la formule qui ne peuvent être toutes simultanément satisfaites.

Définition 15 (Sous-ensemble inconsistant)**.** Soit Φ une formule CNF. Un sous-ensemble ψ de Φ est dit inconsistant ssi il n'existe pas d'interprétation I des variables de Φ permettant de satisfaire toutes les clauses de ψ.

1.2.2 Satisfiabilité propositionelle

Le problème de satisfiabilité propositionnelle, ou problème SAT, est un problème de décision qui peut être défini de la manière suivante.

Définition 16 (Problème SAT)**.** Le problème de satisfiabilité propositionnelle (SAT) consiste à déterminer si une formule propositionnelle en forme normale conjonctive admet un modèle.

Théoreme 1. *Le problème SAT est NP-complet. [Coo71, Lev73]*

Ce problème a suscité beaucoup d'intérêt dans la communauté des chercheurs pour diverses raisons. Tout d'abord, de nombreux problèmes réels peuvent être écrits sous forme de formules propositionnelles, et donc résolus par un solveur SAT. Ensuite, le problème SAT est d'une importance centrale dans le domaine de la théorie de la complexité. Il est le premier problème à avoir été prouvé comme étant NP-complet [Coo71, Lev73]. Un grand nombre de problèmes académiques ont été rattachés à la classe NP-complet par réduction polynomiale avec SAT. De plus, trouver un algorithme polynomial pour résoudre le problème SAT (ou tout autre problème NP-complet) prouverait que P=NP.

Nous présentons dans la suite de ce chapitre les principales règles d'inférence utilisées dans les solveurs SAT avant de donner un rapide aperçu de deux des méthodes de résolution les plus étudiées.

1.2.3 Règles d'inférence pour SAT

Les règles d'inférence permettent de déduire, à partir de la formule initiale, de nouvelles informations ou des simplifications qui peuvent faciliter la résolution du problème.

On peut distinguer deux types de règles d'inférence : les règles syntaxiques qui enrichissent la formule par l'ajout de nouvelles clauses ou la simplification de clauses existantes et les règles sémantiques qui permettent de fixer les valeurs de certaines des variables de l'instance. Nous décrivons ici quelques-unes des règles d'inférence les plus utilisées dans la résolution du problème SAT.

1.2.3.1 Règles d'inférence syntaxiques

Les règles d'inférence syntaxiques modifient la forme même de la formule par l'ajout, la suppression ou la modification de clauses. Nous les décrirons sous la forme :

$$\frac{cp_1, cp_2, \ldots, cp_k}{cc_1, cc_2, \ldots, cc_l}$$

où $cp_1, cp_2, \ldots, cp_k$ sont les clauses prémisses de la règle et $cc_1, cc_2, \ldots, cc_l$ ses conclusions. Pour être valable, une règle d'inférence syntaxique pour SAT doit préserver l'équivalence de la formule. C'est-à-dire que pour toute formule Φ telle que $\{cp_1, cp_2, \ldots, cp_k\} \subset \Phi$, tout modèle de Φ doit être un modèle de $(\Phi \setminus \{cp_1, cp_2, \ldots, cp_k\}) \cup \{cc_1, cc_2, \ldots, cc_l\}$ et inversement.

Suppression des clauses tautologiques Comme nous l'avons vu précédemment, les clauses tautologiques sont satisfaites par toute interprétation complète. Par conséquent, elles n'influent pas sur la satisfiabilité de la formule. On peut donc les supprimer tout en préservant l'équivalence de la formule.

$$\underline{c_i = \{x, \overline{x}, l_1, \ldots, l_k\}} \tag{1.1}$$

Suppression des clauses sous-sommées Considérons deux clauses c_i et c_j telles que c_i sous-somme c_j. Toute interprétation qui satisfait c_i satisfait également c_j. Par conséquent, supprimer c_j ne change pas les modèles de la formule et conserve l'équivalence.

$$\frac{c_i, c_j}{c_i} \tag{1.2}$$

Résolution [Rob65] Une des règles les plus utilisées dans les algorithmes complets pour SAT est la résolution, introduite par Robinson il y a près de cinquante ans. Considérons une formule Φ définie sur un ensemble de variables propositionnelles X et deux clauses $c_j = \{x_i, l_{j_1}, \ldots, l_{j_s}\}$ et $c_k = \{\overline{x_i}, l_{k_1}, \ldots, l_{k_t}\}$ de Φ. Pour que c_j et c_k soient satisfaites par une interprétation I, il faut nécessairement que celle-ci satisfasse $(l_{j_1} \vee \cdots \vee l_{j_s})$ ou $(l_{k_1} \vee \cdots \vee l_{k_t})$. On peut exprimer cette condition en ajoutant une nouvelle clause $cr = \{l_{j_1}, \ldots, l_{j_s}, l_{k_1}, \ldots, l_{k_t}\}$. Nous dirons que la clause cr est le résolvant produit de la résolution entre c_j et c_k sur x_i.

$$\frac{c_i = \{x_i, l_{j_1}, \ldots, l_{j_s}\}, c_j = \{\overline{x_i}, l_{k_1}, \ldots, l_{k_t}\}}{c_i, c_j, cr = \{l_{j_1}, \ldots, l_{j_s}, l_{k_1}, \ldots, l_{k_t}\}} \tag{1.3}$$

La résolution constitue un système complet de réfutation, c'est-à-dire que son application itérative permet de déduire un résolvant vide pour toute formule non-satisfiable.

1.2.3.2 Règles d'inférence sémantiques

Les règles d'inférence sémantiques permettent d'étendre l'interprétation courante en fixant la valeur de variables. Ce faisant, elles éliminent des portions de l'espace de recherche. Par exemple, si une règle fixe la valeur d'une variable x_i à $vrai$, alors toutes les interprétations I telles que $\overline{x_i} \in I$ ne seront plus considérées. Pour être valables, ces règles d'inférence ne doivent pas éliminer de modèle de la formule ou, si elles en éliminent, alors ces modèles doivent être indépendants de la valeur des variables ainsi affectées. Plus formellement, si une règle satisfait un littéral l d'une formule Φ alors il faut que pour tout modèle I de Φ :

— $l \in I$ ou
— $(I \setminus \{\overline{l}\}) \cup \{l\}$ est un modèle de Φ

Propagation unitaire (UP) Une autre règle d'inférence très utilisée dans le cadre de la résolution du problème SAT est la propagation unitaire (*unit propagation*, UP). Considérons une formule Φ définie sur un ensemble de variables propositionnelles X et une clause unitaire $c_j = \{l_i\}$. La clause c_j ne peut être satisfaite que si son unique littéral l_i est lui-même satisfait. On peut donc, dans le cadre du problème SAT, fixer la valeur de $var(l_i)$. Cela revient à supprimer de la formule toutes les occurrences de $\overline{l_i}$ et toutes les clauses contenant l_i. Ce processus peut être répété jusqu'à ce que la formule ne contienne plus aucune clause unitaire ou qu'une clause vide soit générée. Dans ce dernier cas, cela prouve que la formule n'est pas satisfiable. L'ensemble des clauses qui ont conduit, par propagation unitaire, à générer la clause vide constitue alors un sous-ensemble inconsistant de la formule.

Définition 17 (propagation)**.** On appelle *littéraux propagés* (ou simplement *propagations*) les interprétations des variables déduites par propagation unitaire.

Définition 18 (source de propagation)**.** Soit l_i un littéral propagé et I une interprétation partielle. Une clause c_j est une source de propagation de l_i sous l'interprétation I ssi $c_j|_I = \{l_i\}$.

Les étapes de propagation unitaire peuvent être représentées sous forme d'un graphe d'implications qui peut être défini comme suit.

Définition 19 (Graphe d'implications [MSS99])**.** Soit $\Phi = \{c_1, \ldots, c_m\}$ une formule définie sur un ensemble de variables propositionnelles $X = \{x_1, \ldots, x_n\}$ et I une interprétation (partielle ou complète). Nous supposons qu'il ne peut y avoir qu'une seule clause falsifiée (i.e. UP est arrêté dès qu'une clause vide est générée). Un graphe d'implications est un graphe orienté, sans circuit et étiqueté $G = (V, A)$ avec :

$$\begin{aligned} V &= \{l \in I\} \cup \{\Diamond_{c_i} \text{ t.q. } \exists c_i \in \Phi, |c_i| = 1\} \cup \{\Box \text{ si } \exists c_j \in \Phi \text{ falsifiée par } I\} \\ A &= \{(l, l', c_k) \text{ t.q. } l, l' \in I \text{ et } \exists c_k \in \Phi \text{ réduite par } l \text{ et qui propage } l'\} \cup \\ &\quad \{(\Diamond_{c_p}, l, c_p) \text{ t.q. } l \in I \text{ et } \exists c_p = \{l\} \in \Phi\} \cup \\ &\quad \{(l, \Box, c_q) \text{ t.q. } l \in I \text{ et } \exists c_q \in \Phi \text{ falsifié par } I \text{ et } \bar{l} \in c_q\} \end{aligned}$$

Nous utilisons les deux nœuds spéciaux $\Diamond$ et $\Box$ pour représenter respectivement les nœuds initiaux des clauses unitaires de Φ et le nœud final de la clause falsifiée. Par souci de clarté, nous cachons les nœuds $\Diamond$ de la représentation graphique des graphes d'implications. Les arcs sont étiquetés par les clauses causant les propagations.

L'ensemble des sommets V est l'ensemble des décisions et des propagations. Les arcs de A relient les littéraux affectés aux littéraux propagés par les clauses qu'ils réduisent. Dans la suite de cette thèse, nous noterons pour tout littéral $l_i \in V$: $pred_G(l_i)$ les prédécesseurs de l_i dans G, $succ_G(l_i)$ ses successeurs et $neigh_G(l_i)$ son voisinage qui peut être défini par $neigh_G(l_i) = pred_G(l_i) \cup succ_G(l_i)$. Ces notations peuvent être étendues à tout sous-ensemble V' de V comme suit :

$$pred_G(V') = \bigcup_{l_i \in V'} pred_G(l_i) \ \setminus V'$$

$$succ_G(V') = \bigcup_{l_i \in V'} succ_G(l_i) \ \setminus V'$$

$$neigh_G(V') = \bigcup_{l_i \in V'} neigh_G(l_i) \ \setminus V'$$

Nous désignerons par $src_G(l_i)$ la source de propagation d'un littéral propagé l_i dans G et par $red_G(l_i)$ l'ensemble des clauses réduites par l_i dans G.

Règle des littéraux purs [BR99, BF98] La dernière règle d'inférence pour SAT que nous présentons ici est celle des littéraux purs. Si une variable apparaît uniquement positivement (resp. négativement) dans les clauses, alors sa valeur peut être fixée à $vrai$ (resp. $faux$). Cela n'a pas d'impact sur les autres clauses de la formule ni sur la recherche de modèle.

1.3 Méthodes de résolution pour SAT

Depuis les années cinquante, de nombreuses méthodes ont été proposées pour résoudre le problème SAT. La définition au début des années 1990 d'un format unifié pour représenter les formules CNF (le format DIMACS [1]) et l'organisation de compétitions visant à comparer les méthodes de résolution et les implémentations (le second *DIMACS challenge* en 1992-93 [2], puis les *SAT Competitions*, *SAT-Race* et *SAT Challenge* de 2002 à aujourd'hui [3]) ont encouragé le partage des instances et des solveurs. Cela a contribué à stimuler la recherche autour du problème SAT ainsi que le développement de nouvelles techniques de résolution. On peut distinguer deux catégories de solveurs.

Les algorithmes complets (ou systématiques) parcourent l'ensemble de l'espace de recherche des formules qu'ils traitent. Ces algorithmes garantissent de trouver en un temps fini une solution (un modèle) s'il en existe et ils peuvent également prouver qu'une formule n'admet pas de solution. Cependant, le temps et l'espace mémoire nécessaires pour atteindre une solution (ou prouver son absence) peuvent être importants. Pour une formule contenant n variables, il y a 2^n interprétations différentes possibles. Par conséquent, l'exploration complète de l'espace de recherche est très coûteuse. L'efficacité des algorithmes complets dépend donc de leur capacité à ignorer les portions de l'espace de recherche qui ne peuvent contenir de solution. Les algorithmes complets pour SAT reposent sur des méthodes diverses : résolution [Rob65, DP60, Gal77], diagramme de décision binaire [Ake78, Bry92, US94], énumération des solutions [Qui50, DLL62, JW90, MSS99, ES03, SE08], etc.

Les algorithmes incomplets, quant à eux, ne font pas un parcours systématique de l'espace de recherche. Ils se focalisent sur certaines portions de cet espace pour trouver une solution. Par conséquent, ils sont incapables de prouver qu'une formule n'a pas de modèle. Néanmoins, ces solveurs s'avèrent efficaces sur certains groupes d'instances où ils atteignent des solutions plus rapidement que les solveurs complets. Les algorithmes incomplets sont majoritairement basés sur la recherche locale [SLM92, HS04, LH05b], mais on peut aussi citer les algorithmes évolutionnistes [Hol75, Hol92] ou le recuit simulé [KGV83, Cer85].

1. Une description de ce format est disponible à l'adresse `http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps`
2. Pour plus de détails, voir `http://dimacs.rutgers.edu/Challenges/`.
3. Pour plus de détails, voir `http://www.satcompetition.org/`.

Nous présentons dans la suite de cette section deux familles d'algorithmes et certains de leurs composants : l'algorithme DPLL [DLL62] et la recherche locale. Notre but ici n'est pas de fournir une description exhaustive des méthodes de résolution pour SAT, mais plutôt d'introduire certaines techniques et méthodes qui peuvent être utilisées ou adaptées dans le cadre de la résolution du problème Max-SAT.

1.3.1 Solveurs DPLL

Nous présentons ici l'algorithme DPLL, introduit par Davis, Logemann et Loveland en 1962 [DLL62] et dont l'architecture générale est encore utilisée dans un grand nombre des solveurs SAT complets les plus performants.

L'algorithme DPLL explore l'espace des interprétations en faisant une série d'hypothèses (aussi appelées décisions) sur la valeur des variables. Lorsque ces hypothèses falsifient la formule, l'algorithme effectue un retour-arrière. Il défait alors les dernières hypothèses effectuées jusqu'à trouver une interprétation partielle non considérée jusqu'alors. Ce processus est répété jusqu'à ce qu'un modèle de la formule soit trouvé ou que l'ensemble de l'espace de recherche ait été exploré. Le parcours ainsi réalisé de l'espace des interprétations peut être représenté sous la forme d'un arbre binaire communément appelé *arbre de recherche*. À chaque nœud de cet arbre correspond une interprétation partielle des variables de la formule, le nœud initial (la racine) étant l'interprétation vide et les nœuds terminaux (les feuilles) étant les interprétations. Nous utiliserons le vocabulaire suivant dans la suite de cette thèse.

Définition 20 (Décision)**.** On appelle *décisions* ou *branchements* les affectations des variables choisies par l'algorithme DPLL (i.e. les hypothèses) pour les différencier des affectations faites par les règles d'inférence (propagation unitaire et règle des littéraux purs).

Définition 21 (Interprétation courante)**.** A un nœud donné de l'arbre de recherche, on appelle *interprétation courante* l'ensemble des affectations des variables (décisions et propagations) réalisées par l'algorithme.

Définition 22 (Niveau d'une affectation)**.** On définit le niveau d'une affectation (décision ou propagation) faite à un nœud k de l'arbre de recherche comme le nombre de nœuds de décisions présents dans le chemin entre la racine et k.

Définition 23 (Sous-arbre)**.** Lors du parcours de l'arbre de recherche, on appellera *sous-arbre* la sous-partie de l'arbre de recherche enracinée au nœud courant.

L'algorithme 1.1 montre le fonctionnement d'un solveur de type DPLL, qui prend en argument une formule CNF Φ définie sur un ensemble de variables propositionnelles X et retourne un modèle de Φ ou UNSAT si la formule n'admet pas de solution. Nous présentons ici une variante itérative de cet algorithme plutôt que la variante récursive la plus commune. Cela permet à nos yeux de mieux mettre en valeur la manière dont l'arbre de recherche est parcouru et en particulier le système des retours arrières. De plus, cela rend plus visible les similitudes structurelles avec les algorithmes de type séparation et évaluation pour Max-SAT qui seront présentés dans la section 2.3. L'algorithme initialise tout d'abord l'interprétation courante I et le niveau dans l'arbre de recherche (lignes 2-3). Puis il applique des règles d'inférence pour simplifier la formule et étendre l'interprétation

courante (ligne 5). Si une des clauses de la formule est falsifiée par l'interprétation courante, alors l'algorithme effectue un retour-arrière (*backtrack*) dans l'arbre de recherche (lignes 6-7). Sinon, si toutes les variables sont affectées, l'interprétation courante est un modèle de la formule et l'algorithme renvoie ce modèle (lignes 8-9). Sinon, l'algorithme choisi un littéral non affecté l de la formule selon une heuristique de branchement donnée et l'ajoute à l'interprétation courante (lignes 11-14). Ces étapes sont répétées jusqu'à ce qu'un modèle soit trouvé ou que l'ensemble de l'arbre de recherche ait été examiné. Dans ce dernier cas, l'algorithme renvoie UNSAT (ligne 16).

Algorithme 1.1 : Schéma général d'un algorithme de type DPLL

Data : Une formule CNF Φ définie sur un ensemble de variables X.
Result : Un modèle de Φ s'il en existe et UNSAT sinon.

```
1  begin
2      I ← ∅;
3      niveau ← 0;
4      do
5          (Φ, I) ←règles_inférence(Φ, I);
6          if □ ∈ Φ|_I then
7              (niveau, I) ←retour_arrière(restauration, niveau);
8          else if |I| = |X| then
9              return I
10         else
11             l ←heuristique_branchement(Φ|_I);
12             restauration[niveau] ← (faux, I, l);
13             niveau ← niveau + 1;
14             I ← I ∪ {l};
15     while niveau > 0;
16     return UNSAT
```

Algorithme 1.2 : Algorithme DPLL : fonction `retour_arrière`

Data : Un tableau *restauration* de tuples et un entier *niveau*.
Result : Un tuple $(niveau, I)$ avec *niveau* le niveau dans l'arbre de recherche et I l'interprétation courante obtenue après le retour-arrière

```
1  begin
2      repeat
3          niveau ← niveau − 1;
4          if niveau < 0 then return (−1, ∅);
5          (tested, I_d, I_p) ← restauration[niveau];
6      until tested = faux;
7      restauration[niveau] ← (vrai, I, l);
8      I ← I ∪ {l̄};
9      niveau ← niveau + 1;
10     return (niveau, I)
```

Pour assurer le parcours de l'arbre de recherche, l'algorithme 1.1 utilise une liste *restauration* qui contient pour chaque niveau de l'arbre de recherche un tuple $(tested, I,$

l). La première valeur *tested* est *vrai* si les deux branches du niveau de décisions ont été explorées et $faux$ sinon. I est l'interprétation courante au niveau sauvegardé et l est le littéral satisfait par décision. Cette liste est remplie à chaque nouvelle décision (ligne 12) puis utilisé lors des retours arrières comme le montre l'algorithme 1.2 : les interprétations faites à chaque niveau de l'arbre de recherche sont défaites jusqu'à rencontrer une branche inexplorée (lignes 2-6). La valeur de *tested* est alors fixée à *vrai* (ligne 7) tandis que l'interprétation courante et le niveau sont mis à jour puis renvoyés (lignes 8-10).

L'exemple suivant illustre le fonctionnement d'un algorithme de type DPLL.

Exemple 1. Considérons la formule CNF $\Phi = \{c_1, \dots c_4\}$ définie sur un ensemble de variables $X = \{x_1, \dots, x_4\}$ avec $c_1 = \{\overline{x_2}, x_3\}$, $c_2 = \{x_2, x_3\}$, $c_3 = \{\overline{x_1}, \overline{x_3}, x_4\}$ et $c_4 = \{\overline{x_3}, \overline{x_4}\}$. L'application de l'algorithme DPLL sur Φ (en supposant que les branchements sont faits dans l'ordre lexicographique ($x_1 < x_2 < \dots < x_4$) et que la seule règle d'inférence utilisée est la propagation unitaire) se déroule comme suit (l'arbre de recherche est présenté dans la figure 1.2).

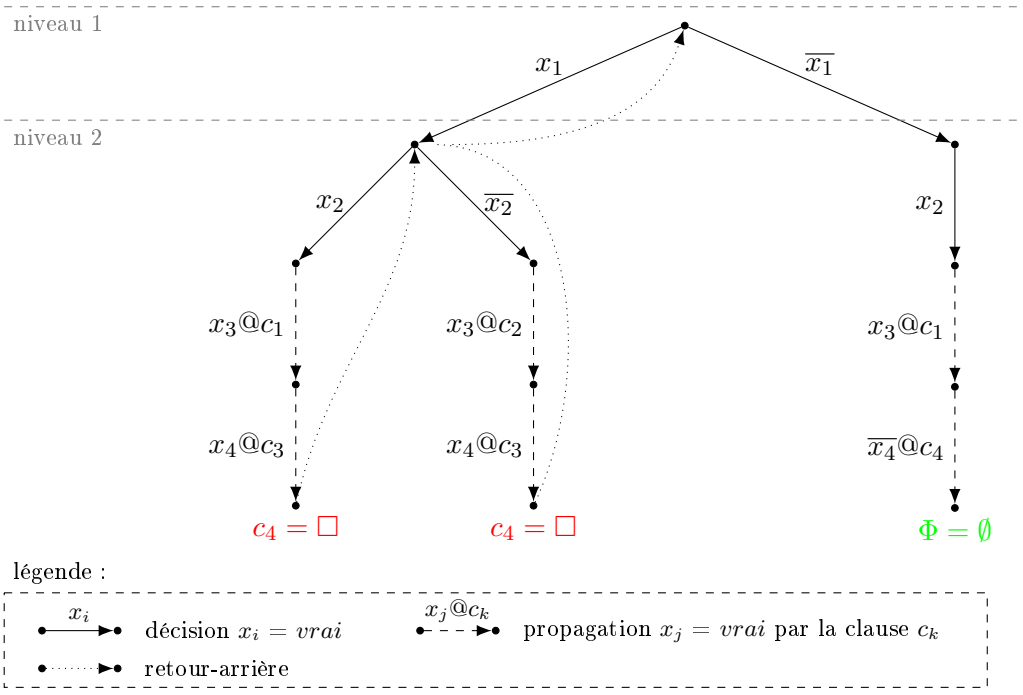


FIGURE 1.2 – Arbre de recherche exploré par un solveur DPLL sur la formule Φ de l'exemple 1. Les branchements sont faits dans l'ordre lexicographique et la seule règle d'inférence utilisée est la propagation unitaire.

La séquence de décisions $\langle x_1, x_2 \rangle$ (i.e. x_1 est affectée à *vrai*, puis x_2 à *vrai*) est réalisée aux deux premiers niveaux de l'arbre de recherche. La clause c_1 est alors unitaire et la propagation unitaire permet d'obtenir la séquence de propagations $\langle x_3@c_1, x_4@c_3 \rangle$ (i.e. x_3 est affectée à *vrai* grâce à la clause unitaire c_1 puis x_4 est affectée à *vrai* par c_3). On a alors l'interprétation courante $I = \{x_1, x_2, x_3, x_4\}$ et la clause c_4 est falsifiée par I. L'algorithme effectue un retour-arrière : il défait les affectations de x_4, x_3 et x_2 et affecte à cette dernière variable la valeur $faux$.

L'application de la propagation unitaire produit la séquence de propagations $\langle x_3@c_2, x_4@c_3\rangle$. L'interprétation courante est alors $I = \{x_1, \overline{x_2}, x_3, x_4\}$ et la clause c_4 est à nouveau falsifiée. L'algorithme effectue un retour-arrière et défait les affectations de x_4, x_3 et x_2. Comme les deux valeurs possibles de x_2 ont déjà été testées, il remonte au niveau précédent de l'arbre de recherche, défait l'affectation de x_1 et lui donne la valeur $faux$.

La variable x_2 est ensuite affectée à *vrai* par décision et la séquence de propagations $\langle x_4@c_1, \overline{x_5}@c_4\rangle$ est réalisée. L'interprétation courante $I = \{\overline{x_1}, x_2, x_3, \overline{x_4}\}$ satisfait toutes les clauses de la formule : un modèle a donc été trouvé.

Les solveurs de type DPLL diffèrent principalement par (1) les règles d'inférence utilisées à chaque nœud de l'arbre de recherche pour simplifier la formule et étendre l'interprétation courante et (2) l'heuristique de branchement qui permet de choisir la prochaine variable à interpréter. Nous décrivons brièvement dans la suite de cette section quelques unes des méthodes utilisées dans ces deux composants avant de décrire les principaux éléments des solveurs *conflict driven clause learning* (CDCL) [MSS99], qui peuvent être vus comme une évolution des solveurs DPLL.

1.3.1.1 Heuristiques de branchement

On distingue trois grandes familles d'heuristiques de branchement, selon les informations qu'elles utilisent pour choisir les variables à affecter.

La première de ces familles est composée des heuristiques dites "syntaxiques", qui se basent sur les apparitions des variables dans les clauses pour évaluer l'impact de leur affectation. Ces heuristiques peuvent être statiques ou dynamiques selon qu'elles prennent en compte les apparitions dans la formule initiale (qui peuvent être calculées une fois pour toutes au début de la recherche) ou les apparitions dans la formule simplifiée par l'interprétation courante (qui doivent être recalculées à chaque nœud de l'arbre de recherche). Parmi les principales heuristiques syntaxiques, on peut citer :

— BOHMS [BB92] : À chaque variable x_i est associé un vecteur $H_1(x_i), H_2(x_i), \ldots, H_n(x_i)$ avec $\forall j \in [1, n]$:

$$H_j(x_i) = \alpha * max(app_j(x_i), app_j(\overline{x_i})) + \beta * min(app_j(x_i), app_j(\overline{x_i}))$$

où $app_j(l)$ est le nombre d'apparitions du littéral l dans les clauses de taille j et α et β sont des constantes fixées empiriquement. La variable choisie par l'heuristique est celle de plus fort vecteur (avec les composantes prises dans l'ordre lexicographique).

— MOMS (*Max number of Occurences in Minimum Size clauses*) [Gol79] : À chaque variable x_i est attribué un score qui peut être défini comme :

$$score_{MOMS}(x_i) = \alpha * app_{min}(x_i) * app_{min}(\overline{x_i}) + app_{min}(x_i) + app_{min}(\overline{x_i})$$

avec α une constante fixée empiriquement et $app_{min}(l)$ le nombre d'apparitions d'un littéral l dans les clauses de petite taille de la formule. Le littéral choisi par l'heuristique est celui de plus fort score. MOMS a inspiré de nombreuses heuristiques de branchement, pour SAT comme pour Max-SAT.

— JW (*Jeroslow-Wang*) [JW90] : A chaque littéral l_i est associé un score défini comme :

$$score_{JW}(l_i) = \sum_{c_j \in \Phi \text{ t.q. } l_i \in c_j} 2^{-|c_j|}$$

Dans la version *one sided* de l'heuristique JW (JW-OS), la variable choisie est celle qui maximise le score :

$$score_{JW-OS}(x_i) = max(score_{JW}(x_i), score_{JW}(\overline{x_i}))$$

et dans la version *two sided* (JW-TS) celle qui maximise le score :

$$score_{JW-TS}(x_i) = score_{JW}(x_i) + score_{JW}(\overline{x_i})$$

On peut remarquer que ces heuristiques tendent à favoriser les variables apparaissant dans les clauses de petite taille, qui sont donc plus à même d'être utilisées dans la propagation unitaire. De plus, ces heuristiques intègrent des mécanismes visant à équilibrer l'arbre de recherche en privilégiant les variables ayant des scores élevés pour les deux polarités.

La seconde famille regroupe les heuristiques de branchement de type *look-ahead*. Ces heuristiques basent leurs choix sur l'estimation de l'impact de l'affectation des variables sur la suite de la recherche. Pour ce faire, elles utilisent par exemple la propagation unitaire [Pre93, Fre95, LA97] ou la probabilité que les variables appartiennent au *backbone*[4] de l'instance [DD04]. Ces méthodes donnent généralement des choix de variable plus précis (au regard de la taille des sous-arbres générés) mais au prix d'un temps de calcul plus long à chaque nœud de l'arbre de recherche.

Enfin, la troisième famille est composée des heuristiques de type *look-back*. À l'inverse des précédentes, elles se basent sur l'expérience accumulée lors des étapes précédentes de la recherche pour évaluer l'intérêt des variables. Elles considèrent en particulier les apparitions des clauses ou des variables dans les conflits (cf. section 1.3.1.2). Ainsi, Brisoux *et al.* [BGS99] ont proposé d'associer à chaque clause un poids selon sa participation dans les conflits. Ce poids peut ensuite être utilisé en combinaison avec n'importe quelle heuristique syntaxique (e.g. MOMS, JW) pour favoriser la détection des conflits dans le haut de l'arbre de recherche. Plus récemment, l'heuristique VSIDS (*Variable State Independent Decaying Sum*) [MMZ+01] associe un poids (nommé "activité") à chaque variable. Lors de l'analyse d'un conflit, les poids des variables qui y participent sont augmentés. Régulièrement, tous les poids sont réduits d'un certain facteur pour privilégier les variables apparaissant dans les conflits découverts récemment. VSIDS choisit à chaque nœud la variable de plus fort poids.

1.3.1.2 Analyse de conflits et apprentissage de clauses

Nous décrivons dans cette section l'apprentissage de clauses par l'analyse de conflits, qui peut être vue comme une utilisation combinée de la propagation unitaire et de la résolution.

Un conflit est détecté lorsqu'à un nœud de l'arbre de recherche la propagation unitaire produit une clause vide (une clause dont tous les littéraux sont falsifiés par l'interprétation

4. Le *backbone* d'une instance SAT est l'ensemble des littéraux satisfaits par tous les modèles de l'instance [MZK+99, BHvMW09].

courante). On peut alors chercher à identifier les raisons du conflit, c'est-à-dire un ensemble d'affectations qui l'ont provoqué. Considérons le graphe d'implications $G = (V, A)$ décrivant les étapes de propagations ayant menées au conflit. Tout ensemble de littéraux $\lambda = \{l_1, \ldots, l_k\} \in V$ tel que λ soit un séparateur de G et que la partie contenant le conflit (le nœud $\square$) soit isolée des décisions (les nœuds cerclés de G) est une raison du conflit. Toute interprétation satisfaisant les littéraux de λ produira, par propagation unitaire, le même conflit. On peut donc ajouter à la formule une clause $c_a = \{\overline{l_1}, \ldots, \overline{l_k}\}$ (dite *clause apprise* ou *nogood*) pour empêcher la satisfaction simultanée des littéraux de λ.

Exemple 2. Considérons le premier conflit détecté par l'algorithme DPLL sur la formule Φ dans l'exemple 1. Pour mémoire, l'interprétation courante est $I = \{x_1, x_2, x_3, x_4, x_5\}$. La clause c_4 est falsifiée. Le graphe d'implications représentant les étapes de propagations unitaires réalisées est donné dans la figure 1.3.

Il y a quatre séparateurs permettant d'isoler les racines du graphe (les décisions) de la feuille (la clause vide $\square$) : $\lambda_1 = \{x_4, x_5\}$, $\lambda_2 = \{x_1, x_4\}$, $\lambda_3 = \{x_3, x_5\}$ et $\lambda_4 = \{x_1, x_3\}$. Chacun de ces séparateurs fournit une explication au conflit, c'est-à-dire une interprétation partielle qui rend la formule inconsistante.

On peut générer pour chacun de ces séparateurs une clause empêchant l'affectation simultanée de ses littéraux. Par exemple, au séparateur λ_4 correspond la clause apprise $c_5 = \{\overline{x_1}, \overline{x_3}\}$. Ajouter cette clause à la formule permet de détecter plus tôt le conflit et évite les redondances dans l'exploration de l'arbre de recherche.

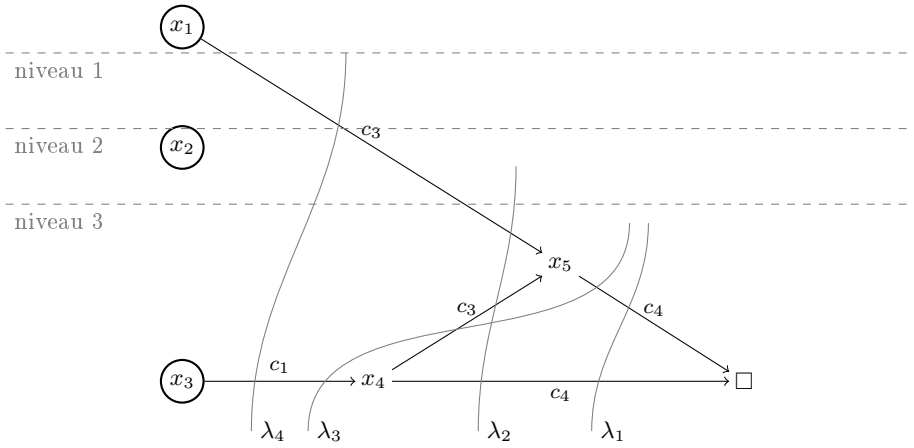


FIGURE 1.3 – Graphe d'implications décrivant les étapes de propagations ayant permis la détection du premier conflit dans l'exemple 1. Les nœuds cerclés sont les décisions et les nœuds non cerclés les propagations. Les étiquettes des arcs sont les clauses provoquant les propagations.

De manière plus formelle, l'ensemble des clauses situées dans le graphe d'implications G entre un séparateur λ et la clause vide est un sous-ensemble inconsistant de la formule $\Phi|_\lambda$. On peut donc fournir une preuve par résolution de la validité de la clause apprise en appliquant itérativement la règle de la résolution (cf. section 1.2.3) entre les clauses du sous-ensemble inconsistant prises dans l'ordre inverse des propagations. Le résolvant final obtenu est la clause apprise.

Exemple 3. Considérons à nouveau le premier conflit détecté par l'algorithme DPLL sur la formule Φ dans l'exemple 1 et le séparateur λ_4 du graphe d'implications présenté dans l'exemple 2. Les clauses situées entre les littéraux de λ_4 et la clause vide $\square$ dans le graphe d'implications sont c_1,c_3 et c_4. La clause apprise peut être déduite en appliquant la règle de la résolution entre ces clauses dans l'ordre inverse des étapes de propagation. La figure 1.4 montre les différentes étapes de ce processus.

La résolution est tout d'abord appliquée entre les clauses c_4 et c_3 sur la variable x_5, produisant le résolvant intermédiaire $cr_1 = \{\overline{x_1}, \overline{x_4}\}$. Puis elle est appliquée entre cr_1 et c_1 sur la variable x_4, produisant le résolvant final (la clause apprise) $cr_2 = \{\overline{x_1}, \overline{x_3}\}$.

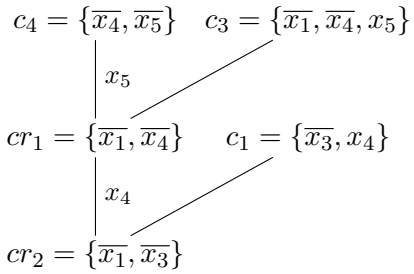


FIGURE 1.4 – Étapes de résolution appliquées lors de la génération de la clause apprise dans l'exemple 3.

1.3.1.3 Solveurs modernes

On appelle solveurs modernes ou solveurs CDCL (pour *Conflict Driven Clause Learning*) [MMZ+01, ES03] une classe de solveurs mixant subtilement des méthodes de résolution sémantiques et syntaxiques. Ces solveurs partagent l'architecture générale des DPLL : ils explorent l'arbre de recherche en faisant des hypothèses sur la valeur des variables (les décisions). En parallèle, ils utilisent l'apprentissage de clauses pour limiter la redondance dans l'exploration de l'arbre de recherche tout en construisant une preuve de réfutation par résolution.

L'interaction entre ces deux composants est forte. L'exploration de l'arbre de recherche conditionne les conflits trouvés et donc les clauses apprises. Inversement, l'analyse de conflits guide l'exploration de l'arbre de recherche à travers :

— des heuristiques de branchement basées sur l'apparition des variables dans les conflits comme VSIDS (cf. section 1.3.1.1 ou [MMZ+01]) et

— les sauts-arrières (*backjump*, ou retours-arrières non-chronologiques) qui utilisent les informations des clauses apprises pour déterminer la profondeur du retour-arrière dans l'arbre de recherche.

Outre ces éléments, les principales particularités des solveurs CDCL par rapport à l'algorithme DPLL sont les suivantes. Ils utilisent un système de relances (*restart*), qui consiste à intervalles réguliers à recommencer la recherche à la racine de l'arbre de recherche. À chaque relance, seules certaines des clauses apprises et les valeurs des activités des variables sont conservées. L'objectif est de concentrer la recherche sur les portions les plus prometteuses de l'espace des interprétations tout en limitant l'impact de l'ordre des bran-

chements sur l'efficacité des solveurs. Les solveurs CDCL utilisent aussi des structures de données dites paresseuses (*lazy data structure*) qui permettent de limiter le temps de calcul passé sur les propagations unitaires à actualiser la formule lors de l'affectation et la désaffectation des variables. La vitesse d'exploration de l'arbre de recherche s'en trouve sensiblement améliorée, en particulier sur les formules de très grande taille.

Nous invitons les lecteurs intéressés à se référer au chapitre 4 de *Handbook of Satisfiability* [RM08] ou à l'ouvrage *Problème SAT : progrès et défis* [Sai08] pour de plus amples détails sur le fonctionnement et l'implémentation des solveurs CDCL.

1.3.2 Recherche locale

Les algorithmes de recherche locale parcourent l'espace de recherche de manière non-systématique. Ils partent d'un interprétation complète puis cherchent à l'améliorer, selon une fonction d'évaluation donnée, par une série de réparations (aussi appelés *mouvements* ou *flips*). À chaque étape, ils cherchent un mouvement qui améliore la fonction d'évaluation. Cette phase d'amélioration est appelée *intensification*. Lorsqu'il n'existe pas de tel mouvement on dit qu'un *optimum local* est atteint. La recherche locale entre alors dans une phase de *diversification* qui vise à atteindre une autre partie de l'espace de recherche pour échapper à l'optimum local. Nous définissons maintenant les deux éléments centraux des méthodes de recherche locale : le voisinage des interprétations qui conditionne les mouvements possibles à chaque étape de la recherche et la fonction d'*évaluation* (ou *coût*) qui permet d'estimer la qualité des interprétations au regard de l'objectif à atteindre.

Définition 24 (Flip)**.** Soit $X = \{x_1, \ldots, x_n\}$ un ensemble de variables propositionnelles, I une interprétation complète de X et x_i une variable de X. Flipper la valeur de x_i dans I consiste à inverser sa valeur (de $vrai$ à $faux$ ou de $faux$ à $vrai$). On notera l'interprétation résultante $flip(I, x_i)$.

On appelle voisinage d'une interprétation I l'ensemble des interprétations qui peuvent être atteintes par un flip :

Définition 25 (Voisinage d'une interprétation)**.** Soit $X = \{x_1, \ldots, x_n\}$ un ensemble de variables propositionnelles, $\mathcal{I}$ l'ensemble des interprétations complètes de X et $I \in \mathcal{I}$ une de ces interprétations. On appelle voisinage de I l'ensemble $\mathcal{N}(I)$ des interprétations complètes qui ont un littéral différent de celui de I. Formellement :

$$\mathcal{N}(I) = \{I' \in \mathcal{I} \text{ t.q. } |I \cap I'| = n - 1\}$$

La recherche locale utilise une fonction d'évaluation pour juger de la qualité des interprétations au regard de l'objectif final (satisfaire la formule). Dans le cadre du problème SAT, on évalue généralement la qualité d'une interprétation en fonction du nombre de clauses de la formule qu'elle falsifie. Plus ce nombre est proche de zéro, plus l'interprétation est proche d'un modèle de la formule.

Définition 26 (Fonction d'évaluation)**.** Soit Φ une formule définie sur un ensemble de variables propositionnelles X et I une interprétation complète de X. La fonction d'évaluation (ou fonction coût) peut être définie comme :

$$eval(\Phi, I) = |\{c_i = \square \in \Phi|_I\}|$$

Remarque 2. Toute interprétation I qui satisfait une formule Φ a une valeur de fonction d'évaluation nulle.

Un autre façon d'évaluer l'impact du flip d'une variable x_i sur la qualité de l'interprétation courante est de considérer le nombre de clauses qui seront falsifiées (noté $break(x_i)$) et satisfaites (noté $make(x_i)$) par le flip. On notera $score(x_i) = break(x_i) - make(x_i)$ la différence entre le nombre de clauses falsifiées par I et $flip(I, x_i)$.

Enfin, on appelle optimum local une interprétation telle qu'il n'existe pas de flip permettant d'améliorer la fonction objectif.

Définition 27 (Optimum local et global)**.** Soit Φ une formule définie sur un ensemble de variables propositionnelles X. Une interprétation complète I des variables de X est un optimum ssi tout déplacement vers ses voisins dégrade la fonction objectif. Formellement :

$$I \text{ est un optimum local de } \Phi \iff \forall I' \in \mathcal{N}(I),\ eval(\Phi, I) \leq eval(\Phi, I')$$

Un optimum local I est dit *global* s'il n'existe pas d'interprétation ayant une valeur de fonction d'évaluation plus faible.

Pour être efficace, un algorithme de recherche locale doit être capable d'améliorer rapidement l'interprétation courante quand c'est possible (phase d'intensification) tout en s'échappant rapidement des optima locaux (phase de diversification). Il faut donc trouver un équilibre entre ces deux phases de la recherche.

1.3.2.1 Schéma général

Les algorithmes de recherche locale pour SAT fonctionnent comme décrit dans l'algorithme 1.3. Ils commencent par construire une interprétation complète I aléatoirement (ligne 3). Puis, ils vérifient si I satisfait la formule en entrée (lignes 5-6). Si c'est le cas, alors un modèle est trouvé et ils renvoient l'interprétation satisfaisant la formule. Sinon, ils choisissent une variable de la formule et ils flippent sa valeur dans I (ligne 8). Si aucune solution n'a été trouvée au bout de maxFlips itérations, l'algorithme recommence avec une nouvelle interprétation aléatoire. L'algorithme s'arrête et renvoi INCONNU si aucune solution n'a été trouvée après maxRelances.

Algorithme 1.3 : Schéma général d'un algorithme de Recherche locale pour SAT

Data : Une formule CNF Φ.
Result : Un modèle de Φ ou INCONNU sinon.

```
1 begin
2     for i de 1 à maxRelances do
3         I ←interprétation_complète_aléatoire();
4         for j de 1 à maxFlips do
5             if Φ|_I = ∅ then
6                 return I
7             else
8                 I ←mouvement(Φ,I);
9     return INCONNU
```

Les algorithmes de recherche locale pour SAT diffèrent principalement par la manière dont ils choisissent les mouvements dans l'espace de recherche. Nous présentons dans la suite de cette section les deux principales architectures sur lesquelles sont basées les algorithmes existants : GSAT et WalkSAT. Puis nous présentons trois variantes ou techniques utilisées dans les solveurs de recherche locale de l'état de l'art : la recherche locale dynamique, la recherche tabou et les mécanismes d'adaptation.

1.3.2.2 GSAT

GSAT, introduit par Selman *et al.* [SLM92], est l'un des premiers algorithmes de recherche locale pour SAT. Dans sa version originale (Algorithme 1.4), le mouvement choisi à chaque étape de la recherche est celui qui minimise la valeur de la fonction objectif. On peut noter qu'il n'y a pas deux phases distinctes d'intensification et de diversification, rendant cet algorithme très sensible aux optima locaux.

Algorithme 1.4 : Algorithme GSAT : fonction `mouvement`

Data : Une formule CNF Φ et une interprétation complète I.
Result : Une interprétation voisine de I.

```
1 begin
2     return une interprétation I' ∈ 𝒩(I) t.q. ∀I'' ∈ 𝒩(I), eval(I') ≤ eval(I'')
```

Parmi les algorithmes de recherche locale qui, comme GSAT, considèrent l'ensemble des voisinages possibles lors du choix des mouvements (i.e. toutes les variables peuvent être flippées) on peut citer G2wsat [LH05b], TNM [LWL12], sattime [LL12], gnovelty+ [PTGS08], sparrow [BF10] ou les solveurs basés sur le *configuration checking* [CS11, CS12, CS13].

1.3.2.3 WalkSAT

L'algorithme WalkSAT [SKC94] est une amélioration de GSAT qui introduit le concept de marche aléatoire. L'algorithme 1.5 montre la fonction de sélection des mouvements de WalkSAT. À chaque itération, l'algorithme choisit aléatoirement une clause c

Algorithme 1.5 : Algorithme WalkSAT : fonction `mouvement`

Data : Une formule CNF Φ et une interprétation complète I.
Result : Une interprétation voisine de I.

```
1 begin
2     c ← une clause de Φ falsifiée par I choisie aléatoirement;
3     if ∃x_i ∈ c t.q. break(x_i) = 0 then
4         return flip(I, x_i)
5     else
6         with probabilité p do
7             return flip(I, x_i) avec x_i la variable de c de plus petit break()
8         otherwise
9             return flip(I, x_i) avec x_i une variable de c choisie aléatoirement
```

falsifiée par l'interprétation courante (ligne 2). S'il existe une variable x_i de c telle que $break(x_i)$ est nulle (i.e. aucune nouvelle clause ne sera falsifiée si x_i est flippée), alors elle est flippée (lignes 3-4). Sinon, avec une probabilité p l'algorithme flippe la variable de c de plus petit $break()$ et avec la probabilité $1-p$ il flippe une variable de c choisie aléatoirement (lignes 5-9).

Le mécanisme de diversification utilisé par WalkSAT est double. La marche aléatoire, qui consiste à choisir aléatoirement la variable à flipper (ligne 9), permet de sortir efficacement des optima locaux. De plus, la limitation des mouvements induite par le choix aléatoire d'un clause falsifiée diversifie encore davantage les mouvements réalisés par WalkSAT.

De nombreux solveurs de recherche locale sont basés sur l'architecture de WalkSAT. Ils considèrent uniquement les voisins accessibles en flippant les littéraux qui apparaissent dans un clause falsifiée choisie aléatoirement. Parmi ces solveurs, on peut citer Novelty et ses variantes [MSK97, Hoo99, LH05b], WalkSATlm [CSL13] ou ProbSAT [BF10].

1.3.2.4 Recherche locale dynamique

Afin de mieux diriger la recherche, Morris [Mor93] a proposé d'associer à chaque clause de l'instance un poids qui reflète la difficulté de la clause. Moins la clause est satisfaite dans les optima locaux atteints par le solveur, plus son poids augmente. Ce poids peut ensuite être utilisé dans la fonction objectif du solveur pour satisfaire en priorité les clauses considérées comme les plus difficiles de la formule. Ces poids sont réduits (ou lissés) régulièrement. Plusieurs implémentations de la recherche locale dynamique ont été réalisées, qui diffèrent par la méthode de calcul des poids des clauses et par l'algorithme de recherche locale utilisé. Parmi elles, on peut citer SAPS [HTH02], PAWS [TPBJ04], gnovelty+ [PTGS08] ou sparrow [BF10].

1.3.2.5 La métaheuristique tabou

La recherche tabou est une méthode bien connue dans le domaine de l'optimisation combinatoire [Glo89]. Elle a été introduite dans les solveurs de recherche locale pour SAT par Mazure *et al.* [MSG97]. Elle consiste à interdire le flip des variables récemment flippées (qui deviennent tabou) pour une durée donnée. L'objectif est d'éviter les cycles dans la recherche pour échapper aux optima locaux. Ce mécanisme peut être adouci par un critère d'aspiration [Glo89] qui permet de passer outre le caractère tabou des variables sous certaines conditions, par exemple quand un mouvement tabou permet d'améliorer la meilleure solution déjà trouvée.

Les méthodes basées sur le *configuration checking* [CS11, CS12, CS13] peuvent être vues comme une forme de tabou, où l'interdiction de flipper des variables ne dépend pas de l'âge de leur dernier flip (c'est-à-dire le nombre de flips réalisés depuis que la valeur de la variable a été modifiée) mais de l'évolution des valeurs des variables voisines.

1.3.2.6 Mécanismes d'adaptation

Un grand nombre de solveurs de recherche locale utilisent des paramètres fixés empiriquement, comme les probabilités de mouvement aléatoire dans les algorithmes de type WalkSAT ou la durée du tabou dans les solveurs basés sur ce mécanisme. Il a été montré

que l'efficacité de la recherche locale dépend grandement des valeurs données à ces paramètres qui nécessitent d'être ajustés finement de manière empirique. De plus, les valeurs optimales peuvent varier selon le type d'instances traitées.

Hoos a donc proposé [Hoo02] un mécanisme permettant d'adapter la valeur de la probabilité p de marche aléatoire de WalkSAT. Si le solveur est dans une phase d'intensification (i.e. il n'a pas rencontré d'optimum local depuis un certain temps), alors on peut augmenter p pour améliorer l'intensification de la recherche sur la portion de l'espace de recherche qui parait prometteuse. A l'inverse, si on rencontre souvent des optima locaux, réduire p conduira à une augmentation du nombre de mouvements aléatoires et donc permettra un déplacement plus rapide vers d'autres portions de l'espace de recherche.

Ce mécanisme, limité à l'origine à la probabilité de marche aléatoire dans WalkSat, peut être étendu à la plupart des paramètres et des algorithmes de recherche locale.

1.4 Conclusion

Nous avons présenté dans ce chapitre le contexte dans lequel se situe l'étude du problème Max-SAT. Nous avons introduit la notion de complexité d'un algorithme et présenté les classes de complexité auxquelles appartiennent les problèmes SAT et Max-SAT. Puis, nous avons donné les notions de logique propositionnelle permettant de comprendre les objets considérés par la suite et nous avons présenté le problème SAT. Nous avons également passé en revue les principales règles d'inférence utilisées dans le cadre de la résolution de SAT et nous avons détaillé le fonctionnement des algorithmes de type DPLL et de la recherche locale.

Nous présentons dans le prochain chapitre le problème Max-SAT. Nous donnons sa définition formelle avant de présenter les principales règles d'inférences et méthodes de résolution qui peuvent lui être appliquées.

Chapitre 2

Le problème Max-SAT

Ce chapitre est consacré à la présentation du problème Max-SAT et des méthodes de résolution qui peuvent lui être appliquées. Nous définissons dans la première section le problème Max-SAT et ses variantes Max-SAT valué et Max-SAT partiel. Nous introduisons les notations qui seront utilisées dans la suite de cette thèse et définissons les notions liées à la résolution de Max-SAT. Dans la seconde section, nous étudions l'applicabilité à Max-SAT des règles d'inférence pour SAT avant de présenter celles dédiées à Max-SAT. Les troisième et quatrième sections présentent respectivement les méthodes de résolution de type séparation et évaluation et recherche locale. Nous décrivons les travaux réalisés jusqu'à présent et discutons des perspectives d'amélioration de ces méthodes. Dans la cinquième section, nous passons brièvement en revue les autres méthodes de résolution existantes. Enfin, nous discutons dans la dernière section de l'évaluation expérimentale des méthodes de résolution pour Max-SAT. Nous présentons les ensembles d'instances (benchmarks) utilisés ainsi que les critères habituellement retenus pour comparer les solveurs.

2.1 Définitions et variantes

Lorsqu'une formule CNF n'admet pas de modèle, il est souvent intéressant de déterminer la moins mauvaise interprétation possible de ses variables. Une manière de mesurer la qualité d'une interprétation consiste à considérer le nombre de clauses de la formule qu'elle falsifie, ou, de manière équivalente, qu'elle satisfait. C'est à cette question que cherche à répondre le problème Max-SAT (*maximum satisfiability*), qui peut être définit formellement comme suit :

Définition 28 (Problème Max-SAT)**.** Étant donnée une formule CNF Φ en entrée, le problème Max-SAT consiste à trouver une interprétation I qui maximise (minimise) le nombre de clauses satisfaites (falsifiées).

Théoreme 2. *Le problème Max-SAT est NP-difficile [Pap94].*

Plusieurs variantes de Max-SAT existent, qui ont un pouvoir expressif plus grand et conviennent souvent mieux à la représentation de problèmes réels. Dans le problème Max-SAT partiel, les clauses de la formule Φ en entrée sont divisées en deux catégories : les clauses dures Φ_D (*hard clauses*) qui doivent être toutes satisfaites et les clauses souples Φ_S (*soft clauses*) qui peuvent être falsifiées. L'objectif est de trouver une interprétation satisfaisant toutes les clauses dures et maximisant (minimisant) le nombre de clauses souples satisfaites (falsifiées). Une instance Max-SAT peut-être représentée sous la forme d'une instance Max-SAT partiel dont toutes les clauses sont souples.

Le problème Max-SAT valué associe à chaque clause c de la formule un poids positif noté $poids(c)$. L'objectif est de maximiser (minimiser) la somme des poids des clauses satisfaites (falsifiées). Une instance Max-SAT peut-être représentée sous la forme d'une instance Max-SAT valué dont toutes les clauses sont de même poids. De même, une instance Max-SAT partiel peut être représentée sous la forme d'une instance Max-SAT valué en donnant un poids infini (ou une valeur suffisamment grande) aux clauses dures et un même poids à toutes les clauses souples.

Enfin, le problème Max-SAT partiel valué est la combinaison des deux précédentes variantes. Les clauses sont divisées en deux catégories, les clauses dures et les clauses

souples, ces dernières étant pondérées. L'objectif est de trouver une interpretation qui satisfait toutes les clauses dures et qui maximise (minimise) la somme des poids des clauses souples satisfaites (falsifiées). Cette dernière variante est la plus expressive et permet de représenter n'importe quelle instance des autres variantes de Max-SAT.

L'intérêt porté à l'étude et à la résolution du problème Max-SAT peut s'expliquer en partie par sa proximité avec le problème SAT. D'autres facteurs y ont également contribué. Le problème Max-SAT et ses variantes ont un fort pouvoir expressif puisqu'ils permettent de représenter, en plus des problèmes d'optimisation combinatoire classiques, des problèmes mixant décision et optimisation (avec les clauses dures) et des problèmes d'optimisation avec préférences (avec les clauses pondérées). De nombreux problèmes académiques d'optimisation peuvent être codés sous forme d'instances Max-SAT puis résolus efficacement par des solveurs Max-SAT. C'est le cas notamment des problèmes Max-CUT [SZ05], Max-CLIQUE [CIKM97] et Min-SAT [LMQZ10, Küg12, ZLMA12]. De la même manière, les instances Max-CSP sur des domaines finis [FW92] et la plupart des instances VALUED-CSP [SFV95] peuvent être transformées en instances Max-SAT. Le problème Max-SAT est aussi adapté à la résolution de problèmes réels. Il a ainsi été utilisé dans des domaines variés tels que : le routage [JKS95, XRS02, FM06], la bio-informatique [SBS05, GL12], la synthèse de circuit [GP95], la vérification de l'intégrité de bases de données [ASM85], le diagnostic [DG12], etc. L'étude du problème Max-SAT a donc à la fois un intérêt théorique puisqu'il peut servir de plate-forme de recherche pour un grand nombre de problèmes d'optimisation combinatoire et un intérêt pratique de par le nombre et la diversité de ses applications.

Sauf mention contraire, nous considérerons dans la suite de cette thèse le problème Max-SAT partiel valué, avec le poids des clauses dures fixé à $+\infty$ (à l'exception des exemples ou par soucis de clarté nous utiliserons des formules non-partielles et non-valuées). Dans ce cadre, l'objectif est de minimiser la somme des poids des clauses souples falsifiées tout en satisfaisant les clauses dures. Nous utiliserons les notions et le vocabulaire suivant.

Définition 29 (coût)**.** Soit Φ une formule et I une interprétation (partielle ou complète) des variables de Φ. Nous dirons que le coût de I correspond à la somme des poids des clauses qu'elle falsifie :

$$cout(\Phi, I) = \sum_{c_j \in \Phi,\ c_j|_I = \square} poids(c_j)$$

Définition 30 (Optimum)**.** Soit Φ une formule et $\mathcal{I}$ l'ensembles des interprétations complètes possibles des variables de cette formule. L'optimum de Φ peut être définie comme le plus petit coût des interprétations possibles :

$$optimum(\Phi) = \min(\{cout(\Phi, I), \forall I \in \mathcal{I}\})$$

Définition 31 (Solution)**.** Une interprétation I est une solution d'une formule Φ ssi elle satisfait toute les clauses dures de la formule. Formellement :

$$I \text{ est une solution de } \Phi \iff cout(\Phi, I) < \infty$$

Nous dirons que I est une *solution optimale* de Φ ssi I est une solution et que $cout(\Phi, I) = optimum(\Phi)$.

Définition 32 (Équivalence pour Max-SAT)**.** On dit que deux instances Max-SAT partiel valué Φ et Φ' sont équivalentes pour Max-SAT si et seulement si pour toute interprétation partielle ou complète I on a $cout(\Phi, I) = cout(\Phi', I)$.

Définition 33 (instance max-k-sat)**.** Une instance Φ du problème Max-SAT (ou de ses variantes) est dites max-k-sat si toutes ses clauses sont de taille inférieure ou égale à un entier positif k.

Tout comme pour le problème SAT, on peut diviser les solveurs pour Max-SAT en deux catégories principales : les solveurs complets et les solveurs incomplets. Les premiers donnent des solutions optimales tandis que les seconds, qui sont généralement plus rapides, ne peuvent prouver l'optimalité des solutions qu'ils renvoient. Parmi les solveurs complets, on peut distinguer trois grandes familles qui reposent sur des techniques de résolution totalement différentes : les solveurs de type séparation et évaluation (*branch and bound*) [LMP07, LMMP10, Küg10, HLO07, HLO08], les solveurs basés sur des appels itératifs à un solveur SAT [FM06, HMS11, KZFH12, MML14, MML12, MML13, MDM14, NB14] et ceux basés sur la reformulation en programmation linéaire en nombres entiers (*integer linear programming*, ILP) [Ach09, AG13]. On peut également citer l'existence de solveurs hybrides [DB11, DB13a, DB13b], qui combinent les approches basées sur l'appel itératif à des solveurs SAT et sur la reformulation en ILP. Des algorithmes portfolio ont également été développés [AMS14]. Les solveurs incomplets sont majoritairement basés sur des algorithmes de recherche locale [CLTS14, HJ90, SHS03, YI98] ou sur des méthodes d'approximation [GvHL06, vMvN05].

Comme pour le problème SAT, une compétition (la *Max-SAT Evaluation*, MSE) est organisée chaque année depuis 2006 pour comparer l'efficacité des méthodes de résolution pour Max-SAT [1]. Les instances utilisées lors de ces compétitions se divisent en trois catégories : aléatoires, *crafted* qui résultent généralement de la transformation d'instances d'autres problèmes d'optimisation, et industrielles qui sont issues de problèmes réels. Les résultats des dernières compétitions montrent certaines tendances parmi les familles de solveurs complets. Les solveurs de type séparation et évaluation dominent les solveurs des autres familles sur les instances aléatoires et certaines *crafted*. Les solveurs basés sur l'appel itératif à un solveur SAT dominent sur les instances industrielles et les solveurs basés sur la reformulation en ILP dominent sur les instances *crafted* partielles.

2.2 Règles d'inférence pour Max-SAT

Comme pour le problème SAT, les règles d'inférence syntaxiques pour Max-SAT doivent préserver l'équivalence de la formule (ie. la formule résultante de l'application de la règle doit être équivalente à l'originale). Cependant, la notion de formule équivalente n'est pas la même pour les problèmes SAT et Max-SAT. Contrairement à SAT où la préservation de la satisfiabilité de la formule suffit à rendre une règle applicable, les règles d'inférence pour Max-SAT doivent également préserver la somme des poids des clauses falsifiées par chaque interprétation. En conséquence, la majorité des règles d'inférence syntaxiques utilisées pour SAT (e.g. la résolution) ne peuvent être appliquées directement à Max-SAT (sauf sur les clauses dures des instances Max-SAT partiel).

1. Détails à l'adresse `http://www.maxsat.udl.cat/`

De la même manière, les règles d'inférence sémantiques pour SAT ne sont pas toutes valables pour Max-SAT. Les solutions du problème SAT et les solutions optimales de Max-SAT ne sont pas les mêmes et, par conséquent, les règles d'inférence sémantiques pour SAT peuvent conduire à ignorer des solutions optimales pour Max-SAT en réduisant l'espace de recherche.

Nous discutons dans la suite de cette section de l'applicabilité des règles d'inférence de SAT au problème Max-SAT avant de présenter les règles dédiées à Max-SAT. Nous considérerons dans toutes les règles présentées ci-dessous qu'au moins une des clauses des prémisses des règles est souple. Les règles syntaxiques peuvent être étendues aux formules valuées comme suit. Si m est le poids minimum des clauses des prémisses, m est soustrait des poids des clauses des prémisses (les clauses de poids nul peuvent être supprimées) et les clauses des conclusions prennent le poids m.

2.2.1 Règles d'inférence pour SAT compatibles avec Max-SAT

Règle des littéraux purs (PL) [BR99, BF98] La règle des littéraux purs, qui consiste à affecter à $vrai$ (resp. $faux$) les variables x_i n'apparaissant que positivement (resp. négativement) dans la formule, peut être appliquée au problème Max-SAT. Une telle affectation ne falsifie aucune clause. Par conséquent, pour toute interprétation optimale I telle que $\overline{x_i} \in I$ (resp. $x_i \in I$) l'interprétation $(I \backslash \{\overline{x_i}\}) \cup \{x_i\}$ (resp. $(I \backslash \{x_i\}) \cup \{\overline{x_i}\}$) est également optimale.

Suppression des clauses tautologiques Les clauses tautologiques sont, par définition, satisfaites par toutes les interprétations complètes des variables de la formule dans laquelle elles apparaissent. Par conséquent, elles n'ont pas d'impact sur la recherche d'une solution optimale et on peut les supprimer de la formule.

2.2.2 Règles d'inférence pour SAT incompatibles avec Max-SAT

Suppression des clauses sous-sommées Nous avons vu que les clauses qui sont sous-sommées peuvent être supprimées de la formule dans le cadre du problème SAT. Cependant, cette règle ne préserve pas l'équivalence pour Max-SAT comme le montre l'exemple suivant.

Exemple 4. Soit une formule CNF $\Phi = \{c_1 = \{x_1, \overline{x_2}\}, c_2 = \{x_1, \overline{x_2}, x_3\}\}$. L'application de la suppression des clauses sous-sommées à Φ produit la formule $\Phi' = \{c_2 = \{x_1, \overline{x_2}\}\}$. Or l'interprétation $\{\overline{x_1}, x_2, \overline{x_3}\}$ falsifie deux clauses de Φ mais une seule de Φ'.

Dans le cas des formules valuées, on peut cependant s'affranchir des clauses strictement équivalentes (i.e. qui contiennent les mêmes littéraux) en les fusionnant. Par exemple, on peut modifier les poids de deux clauses $c_1 = \{l_1, \ldots, l_k\}$ et $c_2 = \{l_1, \ldots, l_k\}$ comme suit : $poids(c_1) = poids(c_1) + poids(c_2)$ et $poids(c_2) = 0$. La formule obtenue est équivalente à l'originale.

Propagation unitaire La propagation unitaire est une règle d'inférence puissante et très utilisée dans les méthodes de résolution complètes pour SAT. Dans le cadre du problème Max-SAT, elle peut conduire à des interprétations non optimales en ignorant des portions de l'espace de recherche pouvant contenir l'optimum. L'exemple suivant illustre cette situation.

Exemple 5. Considérons la formule CNF non-valuée $\Phi = \{c_1, c_2, c_3, c_4, c_5\}$ avec $c_1 = \{x_1\}$, $c_2 = \{\overline{x}_1, x_2\}$, $c_3 = \{\overline{x}_1, \overline{x}_2\}$, $c_4 = \{\overline{x}_1, x_3\}$ et $c_5 = \{\overline{x}_1, \overline{x}_3\}$. La valeur de l'optimum de Φ est 1, avec par exemple l'interprétation $I = \{\overline{x_1}, x_2, x_3\}$. L'application de la propagation unitaire conduit à propager x_1 (affecter x_1 à *vrai*) par la clause c_1. On obtient la formule simplifiée $\Phi|_{\{x_1\}} = \{\{x_2\}, \{\overline{x_2}\}, \{x_3\}, \{\overline{x_3}\}\}$. L'optimum (2) sur l'espace de recherche ainsi réduit est supérieur à l'optimum de la formule originale.

La propagation unitaire peut cependant être utilisée dans les solveurs de type séparation et évaluation pour (1) étendre l'interprétation courante dans certains cas et (2) détecter les sous-ensembles inconsistants de la formule dans le cadre du calcul de la borne inférieure. Ces usages sont décrits respectivement dans les sections 2.3.3 et 2.3.4.

Résolution La règle de la résolution, elle aussi très utilisée dans les solveurs SAT complets de type DPLL, n'est pas applicable dans le cadre du problème Max-SAT. Le nombre de clauses satisfaites ou falsifiées par chaque interprétation n'est pas le même après ajout à la formule de la résolvante et la formule obtenue n'est pas équivalente à l'originale. Nous verrons dans la section suivante la règle de la max-résolution [LH05a, HL06, BLM07], qui peut être vue comme l'adaptation de la résolution pour Max-SAT. Cependant, malgré certains points communs, des différences fondamentales existent entre ces deux règles.

2.2.3 Règles d'inférence dédiées à Max-SAT

De nombreuses règles d'inférence ont été proposées pour Max-SAT et nous ne présentons ici que les principales. En particulier, certaines des règles d'inférence utilisées pour le calcul de la borne inférieure dans les solveurs de type séparation et évaluation seront présentées dans la section 2.3.4.

Dominating unit clauses (DUC) [NR00] Si la somme des poids des clauses unitaires dans lesquelles apparaît un littéral l est supérieure à la somme des poids des clauses dans lesquelles son complémentaire apparaît, alors l sera nécessairement *vrai* dans toutes les solutions de la formule. On peut donc affecter l à *vrai* puisque la portion de l'espace de recherche où l est falsifié ne peut pas contenir de solution optimale.

Max-résolution Nous avons vu que la règle de la résolution ne pouvait être utilisée pour Max-SAT car elle ne préserve pas l'équivalence de la formule. Larossa et Heras [LH05a, HL06] et Bonet et al. [BLM07] ont simultanément proposé une adaptation de la résolution à Max-SAT : la max-résolution. Cette règle peut être définie comme suit :

$$\frac{c_i = \{x, y_1, \ldots, y_s\},\ c_j = \{\overline{x}, z_1, \ldots, z_t\}}{cr = \{y_1, \ldots, y_s, z_1, \ldots, z_t\},\ cc_1,\ \ldots,\ cc_t,\ cc_{t+1},\ \ldots,\ cc_{t+s}} \tag{2.1}$$

avec

$$\begin{array}{ll} cc_1 = \{x, y_1, \ldots, y_s, \overline{z}_1, z_2, \ldots, z_t\} & cc_{t+1} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_1, y_2, \ldots, y_s\} \\ cc_2 = \{x, y_1, \ldots, y_s, \overline{z}_2, \ldots, z_t\} & cc_{t+2} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_2, \ldots, y_s\} \\ \vdots & \vdots \\ cc_t = \{x, y_1, \ldots, y_s, \overline{z}_t\} & cc_{t+s} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_s\} \end{array}$$

c_i et c_j sont les clauses originales de la formule, cr est la *clause résolvante* et $cc_1, \ldots, cc_{t+s}$ sont les *clauses de compensation* ajoutées pour préserver l'équivalence de la formule. On peut noter que les clauses originales c_i et c_j sont supprimées de la formule par l'application de la max-résolution.

Propriété 2. La complexité temporelle dans le pire des cas de l'application de la règle de la max-résolution entre deux clauses c_i et c_j est en $\mathcal{O}(k^2)$ avec $k = max(|c_i|, |c_j|)$.

Démonstration. La complexité temporelle de l'application de la max-résolution est liée au nombre de littéraux qui devront être examinés pour produire la clause résolvante et les clauses de compensation. On a ainsi :

$$|cr| = |c_i| + |c_j| - 2 \tag{2.2}$$

$$\begin{aligned} |cc_1| + \cdots + |cc_t| &= \sum_{l \in \{1,\ldots,|c_2|-1\}} |c_1| + l \\ &= |c_j| * \frac{2 * |c_i| + |c_j|}{2} \end{aligned} \tag{2.3}$$

$$\begin{aligned} |cc_{t+1}| + \cdots + |cc_{t+s}| &= \sum_{l \in \{1,\ldots,|c_1|-1\}} |c_2| + l \\ &= |c_i| * \frac{|c_i| + 2 * |c_j|}{2} \end{aligned} \tag{2.4}$$

En développant les équations 2.2, 2.3 et 2.4 on obtient :

$$|cr| + |cc_1| + \cdots + |cc_{t+s}| = \frac{|c_i|^2}{2} + \frac{|c_j|^2}{2} + 2 * |c_i| * |c_j| + |c_i| + |c_j| - 1 \tag{2.5}$$

La complexité temporelle de l'application de la max-résolution est donc bien en $\mathcal{O}(k^2)$ avec $k = max(|c_i|, |c_j|)$. □

De nombreuses règles d'inférence syntaxiques pour Max-SAT ont été introduites, qui sont toutes des cas particuliers de max-résolution ou d'hyper max-résolution (plusieurs étapes de max-résolution). Nous donnons ici les principales :

— Règle des clauses presque identiques (*almost common clauses*, ACC) [BR99] ou résolution des voisins (*neighborhood resolution*) [LH05a] :

$$\frac{\{x, l_1, l_2, \ldots, l_k\}, \{\overline{x}, l_1, l_2, \ldots, l_k\}}{\{l_1, l_2, \ldots, l_k\}} \tag{2.6}$$

— Résolution chaînée (*chained resolution*) [LMP07, LHdG08]

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \ldots, \{\overline{l_{k-1}}, l_k\}, \{\overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \ldots, \{l_{k-1}, \overline{l_k}\}} \tag{2.7}$$

— Résolution cyclique (*cycle resolution*) [LHdG08, LMMP09]

$$\frac{\{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_k}, \overline{l_1}\}}{\begin{array}{c}\{\overline{l_1}\}, \{\overline{l_1}, l_k\}, \{\overline{l_1}, \overline{l_k}\}, \\ \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \{\overline{l_1}, l_2, \overline{l_3}\}, \{l_1, \overline{l_2}, l_3\} \\ \{\overline{l_1}, l_3\}, \{\overline{l_3}, l_4\}, \{\overline{l_1}, l_3, \overline{l_4}\}, \{l_1, \overline{l_3}, l_4\} \\ \vdots \\ \{\overline{l_1}, l_{k-1}\}, \{\overline{l_{k-1}}, l_k\}, \{\overline{l_1}, l_{k-1}, \overline{l_k}\}, \{l_1, \overline{l_{k-1}}, l_k\}\end{array}} \tag{2.8}$$

Les autres règles d'inférence syntaxiques proposées précédemment peuvent, dans la plupart des cas, être formulées comme des cas particuliers ou des combinaisons de ces trois règles. Par exemple, la règle des clauses complémentaires (*complementary unit clauses*) [NR00] est un cas particulier de la règle 2.6 avec $k = 0$ ou de la règle 2.8, là encore avec $k = 0$. Les règles 2-RES et 3-RES [HL06] sont respectivement des cas particuliers des règles 2.8 avec $k = 3$ et 2.7 avec $k = 3$.

2.3 Méthodes de type séparation et évaluation

Une classe importante de méthodes complètes est basée sur l'algorithme de séparation et évaluation (*branch and bound*), originellement introduit par Land et Doig [LD60]. Cet algorithme explore, par un parcours en profondeur d'abord, l'arbre de recherche associé à la formule traitée comme le font les solveurs SAT de type DPLL (cf. section 1.3.1). À chaque nœud, l'algorithme compare le coût de la meilleure solution trouvée jusqu'à présent (la borne supérieure, UB) avec une sous-estimation du coût de la meilleur solution atteignable dans la branche actuelle de l'arbre de recherche (la borne inférieure, LB). Si $LB \geq UB$, alors on ne peut pas améliorer la meilleure solution connue en étendant l'interprétation partielle courante et l'algorithme réalise un retour-arrière (*backtrack*) jusqu'à atteindre une branche non explorée de l'arbre de recherche. Lorsque ce dernier à été entièrement exploré, la borne supérieure est l'optimum de l'instance.

L'efficacité des solveurs séparation et évaluation repose sur leur capacité à réaliser des coupures importantes lors de l'exploration de l'arbre de recherche, c'est-à-dire à ignorer les branches de l'arbre de recherche qui ne peuvent pas contenir d'interprétation ayant un coût inférieure à celui de la meilleure interprétation rencontrée jusqu'à présent. Plusieurs critères peuvent entrer en jeu :

1. la qualité de la borne supérieure qui doit être proche de la valeur de l'optimum de l'instance,
2. la qualité de la borne inférieure qui doit être proche du coût de la meilleure solution accessible dans la branche actuelle de l'arbre de recherche,
3. l'ordre d'exploration de l'arbre de recherche défini par l'heuristique de branchement,
4. les techniques d'apprentissage, qui évitent les redondances et répétitions dans le calcul de la borne inférieure et l'exploration de l'arbre de recherche,
5. les règles d'inférence sémantiques qui permettent de fixer la valeur de certaines variables.

Dans la suite de cette section, nous présenterons une vue d'ensemble de l'architecture des solveurs de type séparation et évaluation pour Max-SAT. Puis, nous étudions chacun des points évoqués ci-dessus en décrivant les techniques et solutions qui ont été proposées par le passé.

2.3.1 Vue d'ensemble

Un solveur séparation et évaluation pour Max-SAT fonctionne comme suit (Algorithme 2.1). Il commence par initialiser la meilleure solution trouvée jusqu'à présent et la borne supérieure à la pire valeur possible : la somme des poids des clauses souples (ligne 2-3). Puis, il explore l'espace des interprétations en construisant un arbre de recherche. A chaque nœud de cet arbre, l'algorithme essaye dans un premier temps de simplifier la formule et d'étendre l'interprétation courante par le biais de règles d'inférence (fonction `règles_inférence()`, ligne 6). Il calcule ensuite la borne inférieure qui est la somme des poids des clauses souples falsifiées par l'interprétation courante I plus une sous-estimation de la somme des poids des clauses qui seront falsifiées si l'on étend cette interprétation (fonction `calcul_borne_inférieure()`, ligne 7). Si $LB \geq UB$, alors il n'est pas possible de trouver une meilleure solution dans cette branche de l'arbre de recherche et l'algorithme effectue un retour-arrière (fonction `retour_arrière()`, ligne 8) : il défait les décisions faites précédemment jusqu'à trouver une branche inexplorée de l'arbre de recherche. Si l'interprétation courante est complète (i.e. une valeur a été donnée à toutes les variables), alors l'algorithme a trouvé une nouvelle meilleure solution et il

Algorithme 2.1 : Schéma général d'un algorithme séparation et évaluation pour Max-SAT

Data : Une formule CNF Φ avec n le nombre de variables de Φ.
Result : (UB, I_{UB}) avec UB l'optimum de l'instance et I_{UB} une solution optimale si la partie dure de l'instance admet un modèle et UNSAT sinon.

```
1  begin
2      UB ← Σ_{c_j ∈ Φ} poids(c_j);
3      I_UB ← ∅;
4      I ← ∅;
5      do
6          (Φ, I) ←règles_inférence(Φ,I) ;
7          LB ← calcul_borne_inférieure(Φ|_I);
8          if LB ≥ UB then retour_arrière();
9          else if |I| = n then
10             UB ← cout(Φ, I);
11             I_UB ← I;
12             retour_arrière();
13         else
14             l ←heuristique_branchement();
15             I ← I ∪ {l};
16     while |I| > 0;
17     if UB < ∞ then return (UB, I_UB);
18     else return UNSAT;
```

actualise UB avant de faire un retour en arrière (lignes 9-12). Sinon, l'algorithme choisit une variable non affectée suivant une heuristique de branchement et lui donne une valeur (fonction `heuristique_branchement()`, lignes 14-15). Ces étapes sont répétées jusqu'à ce que l'ensemble de l'arbre de recherche ait été exploré. Si la meilleure solution trouvée à un coût infini, alors aucune interprétation ne satisfaisant la partie dure de l'instance n'a été trouvée et l'algorithme renvoie UNSAT. Sinon, il renvoi la valeur de l'optimum et l'interprétation correspondante (lignes 17-18). L'exemple suivant illustre le fonctionnement d'un tel solveur.

Exemple 6. Considérons la formule CNF non valuée $\Phi = \{c_1, \dots c_{10}\}$ avec $c_1 = \{x_1\}$, $c_2 = \{\overline{x_1}\}$, $c_3 = \{\overline{x_2}\}$, $c_4 = \{\overline{x_1}, x_2\}$, $c_5 = \{\overline{x_1}, \overline{x_3}\}$, $c_6 = \{\overline{x_2}, \overline{x_3}\}$, $c_7 = \{\overline{x_4}, \overline{x_5}\}$, $c_8\{x_4, x_5\}$, $c_9 = \{x_4, \overline{x_5}\}$ et $c_{10} = \{\overline{x_1}, x_3, \overline{x_4}\}$. L'application d'un algorithme de type séparation et évaluation sur Φ se déroule comme suit. Nous supposons que les branchements sont faits selon l'ordre lexicographique, que la seule règle d'inférence appliquée est la propagation unitaire réelle (qui sera présentée dans la section 2.3.3) et qu'aucune estimation n'est faite lors du calcul de la borne inférieure (i.e. sa valeur est donc le nombre de clauses falsifiées par l'interprétation courante). L'arbre de recherche décrivant ces étapes est présenté dans la figure 2.1.

La borne supérieure est initialisée à la pire valeur possible : le nombre de clauses de la formule, 10. Puis la séquence de décisions $\langle x_1, x_2, x_3, x_4, x_5 \rangle$ est réalisée. La valeur de la borne inférieure à chacun des nœuds est respectivement 1, 2, 4, 4 et 4. L'interprétation est alors complète et une nouvelle meilleure solution est trouvée. La valeur de la borne supérieure est mise à jour et l'algorithme effectue un retour-arrière jusqu'à la dernière branche inexplorée tel que la valeur de la borne inférieure soit inférieure à la nouvelle valeur de la borne supérieure. Il revient donc au niveau 3 de l'arbre de recherche après avoir défait, dans l'ordre, les affectations des variables x_5, x_4 puis x_3.

La séquence de décisions $\langle \overline{x_3}, x_4 \rangle$ est réalisée. La valeur de la borne inférieure est 2 et 3 respectivement. Comme la différence entre les bornes inférieure et supérieure est de 1, l'algorithme peut appliquer la propagation unitaire réelle qui lui permet de propager $x_5 = faux$ grâce à la clause unitaire c_7. Une nouvelle meilleure solution est trouvée et la valeur de la borne supérieure devient 3. L'algorithme effectue un retour-arrière jusqu'au niveau 4 de l'arbre de recherche. Les affectations des variables x_5 et x_4 sont défaites.

La propagation unitaire réelle permet de fixer directement les valeurs des variables non affectées x_4 et x_5. Trois clauses sont alors falsifiées, et l'algorithme effectue un retour-arrière jusqu'au niveau 2, défaisant ainsi les affectations des variables x_5, x_4, x_3 et x_2.

La variable x_2 est affectée à $faux$ et la propagation unitaire réelle conduit à la séquence de propagations $\langle \overline{x_3}@c_5, \overline{x_4}@c_{10}, x_5@c_8 \rangle$. Trois clauses sont à nouveau falsifiées et l'algorithme effectue un retour-arrière jusqu'au niveau 1 de l'arbre de recherche. Toutes les affectations sont défaites.

La séquence de décisions et propagations $\langle \overline{x_1}, x_2, \overline{x_3}@c_6, x_4, \overline{x_5}@c_7 \rangle$ est réalisée. Une nouvelle meilleure solution est trouvée et l'algorithme revient au niveau 2 de l'arbre de recherche en défaisant les affectations de x_5, x_4, x_3 et x_2.

La séquence de décisions et propagations $\langle \overline{x_2}@c_3, x_3, x_4, \overline{x_5}@c_7 \rangle$ est réalisée. Une nouvelle meilleure solution est trouvée et l'algorithme effectue un retour-arrière jusqu'au niveau 0 de l'arbre de recherche. Ce dernier ayant été entièrement exploré, l'interprétation $I = \{\overline{x_1}, \overline{x_2}, x_3, x_4, \overline{x_5}\}$ est une solution optimale de Φ et on a $optimum(\Phi) = 1$.

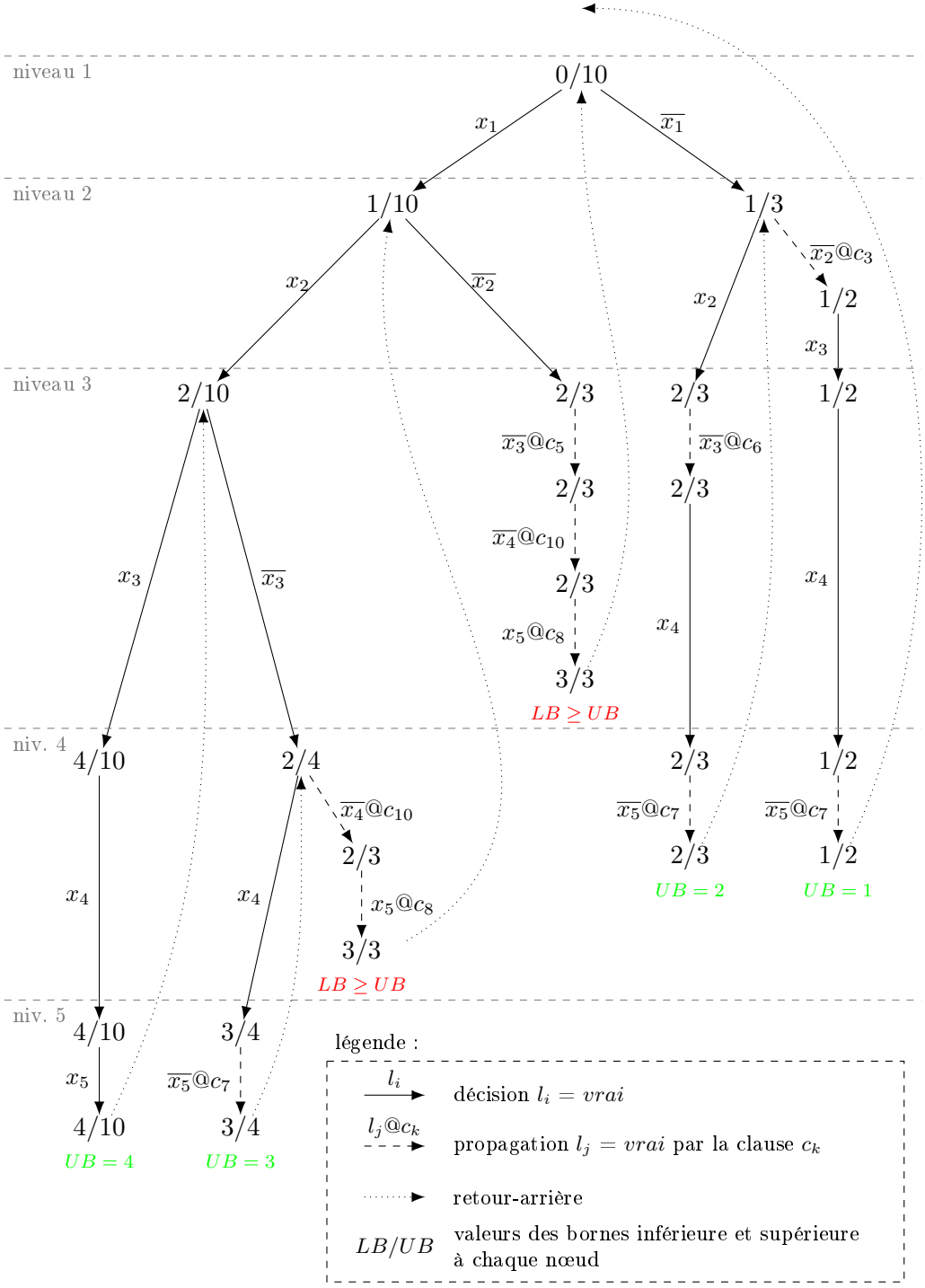


FIGURE 2.1 – Arbre de recherche exploré par l'algorithme séparation et évaluation sur la formule Φ de l'exemple 6. Les branchements sont faits selon l'ordre lexicographique et la seule règle d'inférence utilisée est la propagation unitaire réelle.

2.3.2 Heuristique de branchement

L'heuristique de branchement des solveurs de type séparation et évaluation pour Max-SAT a plusieurs objectifs qui ne sont pas nécessairement compatibles. D'un coté, pour obtenir rapidement une bonne valeur de la borne supérieure, les solveurs doivent choisir en priorité les affectations qui satisfont le plus de clauses possibles. D'un autre coté, pour avoir des valeurs élevées de la borne inférieure dès les premiers nœuds de l'arbre de recherche les solveurs doivent choisir les affectations qui falsifient le plus de clauses possibles. Comme nous le verrons par la suite (cf. section 2.3.4), le calcul de la borne inférieure repose en grande partie sur la propagation unitaire. Par conséquent, les affectations choisies doivent également générer des clauses unitaires et binaires pouvant être utilisées par la propagation unitaire. De plus, comme il est probable que les deux branches soient explorées pour chaque variable choisie, les solveurs doivent choisir les variables permettant d'équilibrer l'arbre de recherche, c'est-à-dire celles qui paraissent le plus à même de donner une bonne estimation de la borne inférieure qu'elles soient affectées à $vrai$ ou à $faux$.

Les premières heuristiques de branchement dédiées aux solveurs Max-SAT [JMB97, AMP03, XZ05] sont des variantes des heuristiques MOMS [Pre93] ou Jeroslow-Wang (JW) [JW90, HV95] pour SAT (cf. section 1.3.1.1). La différence principale entre ces heuristiques et leurs versions originales est que l'importance des clauses unitaires est réduite tandis que celle des clauses binaires est augmentée. Ainsi, ces heuristiques favorisent l'apparition de nouvelles clauses unitaires qui permettent une meilleure estimation de LB par propagation unitaire. Cependant, ces heuristiques cherchent à satisfaire des clauses plutôt qu'à en falsifier. Chaque littéral a un score qui dépend uniquement des clauses dans lesquelles il apparaît. Il n'y a pas de mécanisme pour assurer un équilibre entre les branches de l'arbre de recherche (ie. le littéral de plus fort score est choisi, le score de son complémentaire n'intervient pas dans le choix).

Les solveurs récents utilisent des heuristiques de branchement plus sophistiquées. Par exemple :

— WMAXSATZ [LMP07, LMMP10] donne à chaque littéral l un score basé sur ses apparitions dans les clauses :

$$lscore(l) = app_1(l) + 4 * app_2(l) + app_3(l) \tag{2.9}$$

où $app_k(l)$ est la somme des poids des clauses de taille k dans lesquelles l apparaît. Puis, il choisit la variable x_i de plus fort score selon la formule :

$$score(x_i) = lscore(x_i) * lscore(\overline{x_i}) + lscore(x_i) + lscore(\overline{x_i}) \tag{2.10}$$

On peut remarquer que cette dernière formule, destinée à équilibrer les branches de l'arbre de recherche, est celle introduite par l'heuristique MOMS pour SAT (cf. section 1.3.1.1).

— AKMAXSAT [Küg10] donne lui aussi un score aux littéraux basé sur leurs apparitions dans les clauses :

$$lscore(l) = app_1(l) + 2 * app_2(l) + app_3(l) + app_{4+}(l) \tag{2.11}$$

où $app_k(l)$ est défini comme précédemment pour $k \in \{1, 2, 3\}$ et $app_{4+}(l)$ est la somme des poids des clauses de taille supérieure ou égale à quatre qui contiennent

l. AKMAXSAT prend également en compte un second score $cscore(l)$ basé sur les apparitions des littéraux dans les ensembles inconsistants détectés lors du calcul de la borne inférieure (cf. section 2.3.4). Enfin, il choisit la variable maximisant le score défini comme :

$$score(x_i) = (lscore(x_i) + cscore(x_i)) * (lscore(\overline{x_i}) + cscore(\overline{x_i})) + \min(lscore(x_i), lscore(\overline{x_i})) \quad (2.12)$$

Là encore, la dernière formule vise à maintenir un équilibre entre les branches de l'arbre de recherche en privilégiant les variables ayant de bons scores pour chacune de leurs polarités.

— MINIMAXSAT [HLO07, HLO08] utilise une heuristique de branchement plus originale. Tant que la formule contient des clauses dures, il utilise l'heuristique VSIDS (cf. section 1.3.1.1 ou [MMZ+01]) pour SAT en ne prenant en compte que les clauses dures. Sinon, il utilise une variante valuée de l'heuristique Jerslow-Wang (cf. section 1.3.1.1 ou [HL06]) et il choisit de satisfaire le littéral maximisant :

$$lscore(l) = \sum_{\substack{c_j \in \Phi \text{ t.q.} \\ l \in c_j}} poids(c_j) * 2^{-|c_j|} \quad (2.13)$$

On peut noter qu'il n'y a pas de mécanisme pour équilibrer l'arbre de recherche. C'est à notre connaissance la seule heuristique de branchement traitant séparément les clauses dures et les clauses souples.

2.3.3 Simplifications et affectations par des règles d'inférence

À chaque nœud de l'arbre de recherche, les solveurs de type séparation et évaluation cherchent à simplifier la formule et à étendre l'interprétation courante par le biais de règles d'inférence syntaxiques et sémantiques. Le but des simplifications syntaxiques est de faciliter l'application des règles sémantiques ou d'améliorer la précision de l'estimation de la borne inférieure. Les règles sémantiques quant à elles permettent de fixer la valeur de certaines variables, réduisant ainsi la taille de la sous-partie de l'arbre de recherche restant à explorer.

Règles d'inférence syntaxiques Les principales règles d'inférence syntaxiques utilisées par les solveurs récents (e.g. WMAXSATZ [LMP07], AKMAXSAT [Küg10]) sont les suivantes. Ils fusionnent les clauses identiques, ce qui permet de limiter la fragmentation des informations contenues dans la formule. Ils appliquent également des formes limitées de max-résolution, comme la règle des clauses presque identiques (cf. section 2.2.3) sur les clauses de taille un, deux et parfois trois. Le but de cette dernière règle est de générer de nouvelles clauses unitaires et binaires permettant l'application de règles d'inférence sémantiques comme les littéraux purs ou de détecter plus de sous-ensembles inconsistants par propagation unitaire simulée durant le calcul de la borne inférieure (cf. section 2.3.4).

Règles d'inférence sémantiques Les solveurs séparation et évaluation récents utilisent les règles d'inférence sémantiques compatibles avec Max-SAT présentées en section 2.2 : la règle des littéraux purs et la règle des clauses unitaires dominantes. Cependant, même si ces deux règles permettent de fixer les valeurs de certaines variables, elles

sont appliquées relativement peu souvent en pratique. Leur puissance déductrice est donc faible, en particulier en comparaison de celle de la propagation unitaire utilisée dans le contexte du problème SAT.

Propagation unitaire réelle (HUP) Comme nous l'avons vu précédemment (cf. section 2.2), la règle de la propagation unitaire peut conduire à ignorer des portions de l'espace de recherche contenant une interprétation optimale dans le cadre du problème Max-SAT. Elle peut cependant être utilisée dans certaines situations. Supposons que l'on ait une clause unitaire $c_j = \{l_i\}$ telle que $LB + poids(c_j) \geq UB$. Falsifier le littéral l_i conduira nécessairement à une solution de coût supérieur à celui de la meilleure solution rencontrée jusqu'à présent. Par conséquent, la sous-partie de l'arbre de recherche dans laquelle l_i est falsifié peut être ignorée en satisfaisant l_i. Nous appellerons cette règle *propagation unitaire réelle* (*hard unit propagation*, HUP) pour la différencier de la règle originale ainsi que de la *propagation unitaire simulée* utilisée dans le calcul de la borne inférieure (cf. section 2.3.4).

La capacité de cette règle à interpréter des variables dépend des valeurs des bornes inférieures et supérieures. Elle donne toute sa mesure lorsque la borne inférieure a une valeur proche de celle de la borne supérieure. Lorsque c'est le cas, la propagation unitaire réelle permet généralement de fixer rapidement la valeur des variables non affectées et donc d'invalider la branche courante de l'arbre de recherche ou d'améliorer la borne supérieure. Cela renforce encore l'impact de la qualité des bornes supérieure et inférieure sur l'efficacité des solveurs de type séparation et évaluation.

2.3.4 Calcul de la borne inférieure

À chaque nœud de l'arbre de recherche, les solveurs de type séparation et évaluation calculent la borne inférieure, qui est une sous-estimation du coût de la meilleure solution (autrement dit de la somme minimale des poids des clauses falsifiées) accessible dans la branche courante de l'arbre de recherche. Ce composant a un impact important sur l'efficacité des solveurs pour deux raisons. Tout d'abord, il est appliqué très souvent par conséquent le temps de calcul qui lui est consacré conditionne la vitesse de l'exploration de l'arbre de recherche par les solveurs. Ensuite, la qualité de l'estimation permet de faire des coupures dans l'arbre de recherche, et elle influe donc sur le nombre de nœuds explorés. Pour être efficace, les solveurs doivent donc trouver un équilibre entre le temps passé au calcul de LB et sa qualité.

La forme la plus simple de calcul de la borne inférieure consiste à compter la somme des poids des clauses de la formule Φ qui sont falsifiées par l'interprétation courante I :

$$clauses\_falsifiees(\Phi|_I) = \sum_{c_j \in \Phi,\ c_j|_I = \square} poids(c_j) \tag{2.14}$$

Cette valeur est cependant souvent très éloignée du coût de la meilleur solution possible, en particulier dans les nœuds de l'arbre de recherche proches de la racine. Elle peut être améliorée en estimant le poids des clauses qui seront falsifiées si l'on étend l'interprétation courante.

La méthode la plus courante permettant de réaliser cette estimation consiste, à chaque nœud de l'arbre de recherche, à compter la somme des poids des sous-ensembles inconsistants disjoints présents dans la formule simplifiée par l'interprétation courante. En effet,

au moins une clause de chacun de ces sous-ensembles inconsistants sera falsifiée si l'on étend l'interprétation courante.

On peut distinguer dans cette méthode deux phases distinctes : la détection des sous-ensembles inconsistants et leur traitement, nécessaire pour assurer leur caractère disjoint. Nous présentons dans la suite de cette section les principaux travaux et méthodes existants pour chacune de ces deux phases.

Par soucis d'exhaustivité, il faut mentionner l'existence d'autres méthodes d'estimation de la borne inférieure reposant sur la reformulation en programmation linéaire en nombres entiers [XZ05] ou en problème de cardinalité minimum [PD07, PPC+08].

2.3.4.1 Détection des sous-ensembles inconsistants

Inconsistency count (IC) [WF96] La première méthode permettant de prendre en compte des sous-ensembles inconsistants dans le calcul de la borne inférieure est l'*inconsistency count*, introduite par Wallace et Freuder [WF96]. Elle consiste à considérer, en plus des clauses falsifiées par l'interprétation courante I, les sous-ensembles inconsistants de la forme :

$$\{l_i\}, \{\overline{l_i}\}$$

LB4 [SZ04] Shen et Zhang ont amélioré l'*inconstency count* dans leur méthode LB4 en exploitant les clauses unitaires non utilisées par IC. LB4 utilise une forme limitée de résolution pour générer de nouvelles clauses unitaires. Ainsi, pour deux clauses $\{l_1\}$ et $\{\overline{l_1}, l_2\}$, il génère par résolution la clause résolvante $\{l_2\}$. LB4 peut donc détecter certains sous-ensembles inconsistants de la forme :

$$\{l_1\}, \{\overline{l_1}, l_2\}, \ldots, \{\overline{l_{k-1}}, l_k\}, \{\overline{l_k}\}$$

LB4 utilise un ordre statique des variables. Par conséquent, un tel ensemble inconsistant ne sera détecté que si $\forall i \in \{1, \ldots, k-1\}$, la variable $var(l_i)$ est avant $var(l_{i+1})$ dans l'ordre. On peut également noter que LB4 capture (avec $k = 1$) tous les sous-ensembles inconsistants détectés par IC.

Star rule [NR00, AMP04] La *star rule*, introduite par Niedermier et Rossmanith [NR00] et utilisée pour la première fois dans le cadre du calcul de la borne inférieure par Alsinet *et al.* [AMP04], considère les sous-ensembles inconsistants de la forme :

$$\{l_1\}, \{l_2\}, \ldots, \{l_k\}, \{\overline{l_1}, \overline{l_2}, \ldots, \overline{l_k}\}$$

Dans l'implémentation originale, seuls les cas $k = 1$ et $k = 2$ sont pris en compte. On peut noter que la *star rule* capture tous les sous-ensembles inconsistants détectés par IC (avec $k = 1$) et certains de ceux détectés par LB4.

Propagation unitaire simulée (SUP) [LMP05] Plus récemment, Li *et al.* ont introduit [LMP05] une nouvelle méthode de détection des sous-ensembles inconsistants basée sur la règle de la propagation unitaire (cf. section 1.2.3.2) : la propagation unitaire simulée (*simulated unit propagation*, SUP).

Nous avons vu (cf. section 1.3.1.3) que la propagation unitaire est utilisée dans le cadre des solveurs SAT modernes pour étendre l'interprétation courante et pour détecter

les conflits. Lorsqu'une clause vide est générée par la propagation unitaire, l'ensemble des clauses utilisées dans la séquence de propagations ayant mené au conflit est un sous-ensemble inconsistant de la formule (au nœud actuel de l'arbre de recherche). Ces sous-ensembles inconsistants sont ensuite analysés pour déterminer une clause apprise et le niveau de saut arrière.

Li et al. ont proposé d'utiliser la propagation unitaire de la même manière pour détecter les sous-ensembles inconsistants disjoints de la formule au nœud courant de l'arbre de recherche. L'algorithme 2.2 décrit cette procédure. L'interprétation J, destinée à contenir les propagations simulées, est initialement vide (ligne 2). Tant que la formule contient des clauses unitaires, les littéraux qu'elles contiennent sont satisfaits (lignes 3-4). Si une clause vide est détectée (ligne 5), un sous-ensemble inconsistant ψ est construit en analysant les étapes de propagation ayant menées au conflit (ligne 6). Ce sous-ensemble inconsistant est ensuite transformé pour éviter qu'il ne puisse être compté plusieurs fois dans l'estimation de la borne inférieure (ligne 7). Cette transformation ajoute une clause vide à la formule dont le poids correspond au poids minimal des clauses de ψ. Les méthodes de transformation existantes seront présentées dans la section 2.3.4.2). Comme ces transformations suppriment des clauses de la formule, certaines des propagations simulées faites précedemment ne sont plus valables et doivent être défaites. L'algorithme réinitialise l'interprétation J (ligne 8). Lorsque plus aucune clause unitaire n'est présente dans la formule, l'algorithme renvoie la somme des poids des clauses falsifiées par l'interprétation courante (ligne 9).

Algorithme 2.2 : Fonction `calcul_borne_inférieure` par propagation unitaire simulée.

Data : Une formule CNF Φ et une interprétation partielle I.
Result : La valeur de l'estimation de la borne inférieure.

```
1 begin
2     J ← ∅;
3     while ∃c_i ∈ Φ, c_i|_{I∪J} = {l} do
4         J ← J ∪ {l};
5         while ∃c_j ∈ Φ, c_j|_{I∪J} = □ do
6             ψ ←construction_sous_ensemble_inconsistant(Φ,I ∪ J,c_j);
7             Φ ←transformation_sous_ensemble_inconsistant(Φ,ψ);
8             J ← ∅;
9     return Σ_{c_j∈Φ, c_j|_I=□} poids(c_j);
```

Comme la propagation unitaire peut conduire à ignorer des solutions optimales (cf. section 2.2.2), les affectations faites par propagation unitaire pendant le calcul de la borne inférieure ne sont pas ajoutées à l'interprétation courante I. C'est pourquoi on parle de propagation unitaire simulée. Il est intéressant de noter que cette méthode de détection capture tous les sous-ensembles inconsistants détectés par les méthodes précédentes (IC, LB4 et *star rule*).

Chaque clause souple ne peut participer à des sous-ensembles inconsistants qu'à hauteur de son poids. De plus, les propagations unitaires simulées ne peuvent être réalisées que lorsque la formule contient des clauses unitaires "réelles", c'est-à-dire réduites uniquement par des affectations non simulées (par opposition aux clauses unitaires "simulées"

dont certains littéraux sont falsifiés par des affectations simulées). Plus la formule contient de telles clauses et plus SUP pourra détecter de sous-ensembles inconsistants. En se basant sur ces observations, Li *et al.* ont introduit [LMP06] deux variantes de SUP pour influer sur la structure des sous-ensembles inconsistants et réduire le nombre de clauses unitaires réelles qu'ils contiennent. La première variante, SUP^S, utilise une pile plutôt qu'une queue pour stocker les clauses unitaires. Ainsi, les dernières clauses unitaires ajoutées sont les premières à être traitées. La seconde variante, SUP*, utilise deux queues distinctes : une pour les clauses unitaires réelles et une pour les clauses unitaire simulées. SUP* utilise en priorité les clauses unitaires simulées, puis les clauses unitaires réelles. L'analyse expérimentale [LMP06] réalisée par les auteurs montre que ces deux variantes améliorent la qualité de l'estimation faite par SUP, avec un net avantage pour la variante SUP*.

Exemple 7. Considérons la formule non valuée $\Phi_1 = \{c_1, \ldots, c_5\}$ avec $c_1 = \{x_1\}$, $c_2 = \{\overline{x_1}, x_2\}$, $c_3 = \{\overline{x_2}, x_3\}$, $c_4 = \{\overline{x_3}, x_4\}$ et $c_5 = \{\overline{x_2}, \overline{x_4}\}$. L'application de la propagation unitaire simulée produit la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4\rangle$. La clause c_5 est alors vide. Le graphe d'implications décrivant ces étapes de propagations est présenté dans la figure 2.2. L'ensemble des clauses $\psi = \{c_1, c_2, c_3, c_4, c_5\}$ ayant mené par propagation au conflit constitue un sous-ensemble inconsistant de la formule.

On peut noter qu'aucune des autres méthodes de calcul de la borne inférieure présentées précédemment n'est capable de détecter un tel sous-ensemble inconsistant.

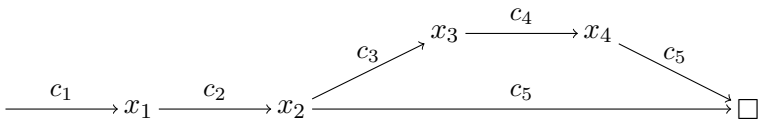


FIGURE 2.2 – Graphe d'implications décrivant les étapes de propagations simulées réalisées sur la formule Φ_1 dans l'exemple 7

.

Littéraux contradictoires (FL) [LMP06] Li *et al.* ont proposé une seconde méthode [LMP06], basée également sur la propagation unitaire, qui complète SUP : les littéraux contradictoires (*failed literals*, FL). Considérons une formule Φ et une interprétation partielle I. La méthode des littéraux contradictoires consiste à tester, pour chaque variable non affectée x_i, si l'application de SUP permet de détecter un sous-ensemble inconsistant ψ dans la formule $\Phi_{I\cup\{x_i\}}$, puis un sous-ensemble inconsistant ψ' dans $\Phi_{I\cup\{\overline{x_i}\}}$. Si c'est le cas, alors $\psi \cup \psi'$ est un sous-ensemble inconsistant de $\Phi|_I$.

L'algorithme 2.3 présente l'utilisation combinée de SUP et FL telle qu'implémentée dans le solveur WMAXSATZ [LMP06, LMP07]. Les lignes 2 à 6 correspondent à l'application de SUP comme détaillé dans l'algorithme 2.2. Les lignes 7-23 correspondent à l'application des littéraux contradictoires. On peut remarquer que, pour limiter le temps de calcul passé à détecter les littéraux contradictoires, seules sont considérées les variables qui peuvent permettre de détecter de nouveaux sous-ensembles inconsistants (ie. qui apparaissent positivement et négativement dans des clauses binaires). De plus, un seul sous-ensemble inconsistant est détecté pour chaque variable considérée.

Algorithme 2.3 : Fonction `calcul_borne_inférieure` par propagation unitaire simulée et technique des littéraux contradictoires.

Data : Une formule CNF Φ et une interprétation partielle I.
Result : La valeur de l'estimation de la borne inférieure.

```
1  begin
2      while ∃c_i = {l} ∈ Φ|_I do
3          I ← I ∪ {l};
4          while ∃c_j = □ ∈ Φ|_I do
5              ψ ←construction_sous_ensemble_inconsistant(Φ,I,c_j);
6              Φ ←transformation_sous_ensemble_inconsistant(Φ,ψ);
7      for x ∈ var(Φ|_I) t.q. app_2(x) > 0 et app_2(x̄) > 0 do
8          I' ← I ∪ {x};
9          ψ ← ∅;
10         while ∃c_i = {l} ∈ Φ|_I' do
11             I' ← I' ∪ {l};
12             if ∃c_j = □ ∈ Φ|_I' then
13                 ψ ←construction_sous_ensemble_inconsistant(Φ,I',c_j);
14                 break;
15         I' ← I ∪ {x̄};
16         ψ' ← ∅;
17         while ∃c_i = {l} ∈ Φ|_I' do
18             I' ← I' ∪ {l};
19             if ∃c_j = □ ∈ Φ|_I' then
20                 ψ' ←construction_sous_ensemble_inconsistant(Φ,I',c_j);
21                 break;
22         if ψ ≠ ∅ et ψ' ≠ ∅ then
23             Φ ←transformation_sous_ensemble_inconsistant(Φ,ψ ∪ ψ');
24     return Σ_{c_j∈Φ, c_j|_I=□} poids(c_j);
```

Cette technique a été améliorée par Kügel [Kü g10] qui regroupe SUP et FL en une seule et unique procédure nommée propagation unitaire généralisée (*generalized unit propagation*, GUP). L'idée sous-jacente à GUP est de mieux utiliser les littéraux contradictoires. Ainsi, si SUP a permis de détecter un sous-ensemble inconsistant ψ dans la formule $\Phi|_{I\cup\{x_i\}}$, on sait que x_i doit être fixé à $faux$ pour éviter qu'une clause de ψ ne soit falsifiée. Si aucun sous-ensemble inconsistant ne peut être détecté dans $\Phi|_{I\cup\{\overline{x_i}\}}$, on peut malgré tout utiliser ψ et continuer à appliquer FL dans la formule $\Phi|_{I\cup\{\overline{x_i}\}}$.

Exemple 8. Considérons la formule non valuée $\Phi_2 = \{c_1, \ldots, c_5\}$ avec $c_1 = \{x_1, x_2\}$, $c_2 = \{\overline{x_2}, x_3\}$, $c_3 = \{x_1, \overline{x_3}\}$, $c_4 = \{\overline{x_1}, x_4\}$ et $c_5 = \{\overline{x_1}, \overline{x_4}\}$. Φ_2 ne contenant aucune clauses unitaire, aucun sous-ensemble inconsistant ne peut être détecté en utilisant la propagation unitaire simulée.

Appliquons la méthode des littéraux contradictoires sur la variable x_1. L'application de SUP sur la formule $\Phi_2|_{\{x_1\}}$ permet d'affecter la variable x_4 à $vrai$ par la clause unitaire c_4. La clause c_5 est falsifiée, par conséquent $\psi_1 = \{c_4, c_5\}$ est un sous-ensemble inconsistant de $\Phi_2|_{\{x_1\}}$.

Si l'on fixe la valeur de x_1 a $faux$, l'application de SUP sur $\Phi_2|_{\{\overline{x_1}\}}$ produit la séquence

de propagations $\langle x_2@c_1, x_3@c_2\rangle$ et la clause c_3 est falsifiée. $\psi_2 = \{c_1, c_2, c_3\}$ est un sous-ensemble inconsistant de $\Phi_2|_{\{\overline{x_1}\}}$.

Donc $\psi_1 \cup \psi_2$ est un sous-ensemble inconsistant de Φ_2.

2.3.4.2 Transformation des sous-ensembles inconsistants

Une fois détectés, les sous-ensembles inconsistants doivent être transformés pour qu'ils ne puissent être comptés qu'une fois dans l'estimation de la borne inférieure. Deux méthodes de transformation sont utilisées par les solveurs existants : la suppression et la transformation par max-résolution.

Suppression des sous-ensembles inconsistants La méthode la plus simple de transformation des sous-ensembles inconsistants consiste à les supprimer de la formule. Ainsi, pour un sous-ensemble inconsistant ψ dont le poids minimal des clauses est m, on peut soustraire m du poids de chaque clause de ψ (les clauses de poids vide sont supprimées de la formule) et ajouter une clause vide de poids m. L'algorithme 2.4 détaille le fonctionnement de cette méthode de transformation.

Algorithme 2.4 : Fonction `transformation_sous_ensemble_inconsistant` par suppression.

Data : Une formule CNF Φ et un sous-ensemble inconsistant ψ.
Result : La formule transformée.

```
1 begin
2     m ← min({poids(c), ∀c ∈ ψ});
3     for c ∈ ψ do
4         poids(c) ← poids(c) − m;
5     poids(□) ← poids(□) + m;
6     return Φ;
```

Cette méthode a plusieurs avantages. Elle est peu coûteuse en temps de calcul et elle n'entraîne pas d'augmentation de la taille de la formule. Cependant, la formule obtenue après son application n'est pas équivalente à l'originale. Elle peut en particulier contenir moins de sous-ensembles inconsistants (et par conséquent avoir une valeur d'optimum plus faible). Pour cette raison, la suppression des sous-ensembles inconsistants est toujours locale au nœud courant de l'arbre de recherche : les changements apportés à la formule sont conservés uniquement pour la durée du calcul de la LB. L'exemple suivant illustre le fonctionnement et les limites de cette méthode.

Exemple 9. Considérons la formule CNF non valuée $\Phi_3 = \{c_1, \ldots, c_8\}$ avec $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x_1}, \overline{x_2}\}$, $c_4 = \{\overline{x_3}, x_4\}$, $c_5 = \{\overline{x_3}, x_5\}$, $c_6 = \{\overline{x_4}, x_6\}$, $c_7 = \{\overline{x_4}, \overline{x_5}, x_7\}$ et $c_8 = \{\overline{x_6}, \overline{x_7}\}$. L'application de SUP* sur Φ_3 produit la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4, x_5@c_5, x_6@c_6, x_7@c_7\rangle$ et la clause c_8 est vide. La figure 2.3 montre le graphe d'implications décrivant les étapes de propagation unitaires réalisées. L'ensemble $\psi = \{c_1, \ldots, c_8\}$ est donc un sous-ensemble inconsistant de la fomule.

L'application de la suppression temporaire sur ψ produit la formule transformée $\Phi'_3 = \{\square\}$. On peut remarquer que Φ'_3 n'est pas équivalente à la formule originale Φ_3. Par

exemple, si l'on considère l'interprétation partielle $I = \{x_1, x_2\}$, on a $cout(\Phi'_3, I) = 1$ et $cout(\Phi_3, I) = 2$.

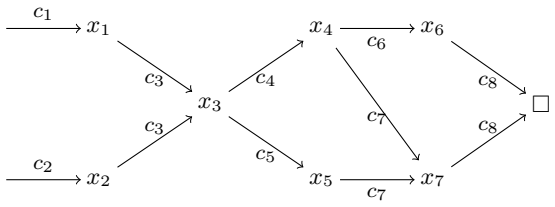


FIGURE 2.3 – Graphe d'implications décrivant les étapes de propagation unitaire simulée appliquées sur la formule Φ_3 dans l'exemple 9.

Transformation par max-résolution L'autre méthode de transformation des sous-ensembles inconsistants est basée sur la règle de la max-résolution (cf. section 2.2.3). Elle consiste à appliquer une série d'étapes de max-résolution entre les clauses des sous-ensembles inconsistants. Si m est le poids minimum des clauses d'un sous-ensemble inconsistant ψ, m est soustrait des clauses originales de ψ. Une clause résolvante (vide si le sous-ensemble inconsistant est entièrement transformé) et des clauses de compensation sont ajoutées à la formule. Toutes les clauses ajoutées prennent le poids m. Les résolvantes intermédiaires générés à chaque étape de max-résolution sont consumés par les étapes suivantes de la transformation.

Algorithme 2.5 : Fonction `transformation_sous_ensemble_inconsistant` par max-résolution.

Data : Une formule Φ, un sous-ensemble inconsistant ψ et un graphe d'implications $G = (V, A)$ décrivant les étapes de propagation unitaire ayant conduit à la détection de ψ.
Result : La formule Φ transformée.

```
1  begin
2      m ← min({poids(c), ∀c ∈ ψ});
3      c1 ← clause falsifiée de ψ;
4      while c1 ≠ □ do
5          x ← la variable de V non traitée propagée la plus récemment;
6          c2 ← src_G(x);
7          (cr, CC) ← max_resolution(Φ,c1,x,c2);
8          poids(c1) ← poids(c1) − m;
9          poids(c2) ← poids(c2) − m;
10         c1 ← cr;
11     return Φ;
```

Cette méthode a l'avantage de préserver l'équivalence de la formule. Elle est cependant coûteuse en temps de calcul en comparaison de la suppression des sous-ensembles inconsistants. Elle peut également entraîner un augmentation de la taille de la formule due à l'ajout des clauses de compensation. À notre connaissance, tous les solveurs utilisant cette méthode de transformation conservent les changements apportés à la formule

dans la sous-partie de l'arbre de recherche. Les clauses résolvantes étant falsifiées par l'interprétation courante, les sous-ensembles inconsistants ainsi transformés sont directement détectables dans la sous-partie de l'arbre de recherche. Cette méthode de transformation agit donc comme une forme (limitée) d'apprentissage et permet de rendre le calcul de la borne inférieure plus incrémental, en réduisant la redondance dans la détection et la transformation des sous-ensembles inconsistants. L'exemple suivant illustre le fonctionnement de cette méthode de transformation.

Exemple 10. Considérons à nouveau la formule Φ_3 de l'exemple 9. Nous avons vu que l'application de SUP* permettait de détecter le sous-ensemble inconsistant $\psi = \{c_1, \ldots, c_8\}$ par la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4, x_5@c_5, x_6@c_6, x_7@c_7\rangle$.

La transformation de ψ par max-résolution est réalisée comme suit. La règle est tout d'abord appliquée entre la clause falsifiée c_8 et la source de propagation c_7 de la dernière variable propagée x_7. Le résolvant $cr = \{\overline{x_4}, \overline{x_5}, \overline{x_6}\}$ ainsi que les clauses de compensation $cc_1 = \{\overline{x_4}, \overline{x_5}, \overline{x_6}\}$, $cc_2 = \{x_4, \overline{x_5}, \overline{x_6}\}$ et $cc_3 = \{x_5, \overline{x_6}\}$ sont ajoutées à la formule. Les clauses originales c_7 et c_8 sont supprimées. Puis, la max-résolution est appliquée entre la clause résolvante cr_1 obtenue précédemment et la source de propagation c_6 de la dernière variable propagée non traitée x_6. La clause résolvante $cr_2 = \{\overline{x_4}, \overline{x_5}\}$ ainsi que la clause de compensation $cc_4 = \{\overline{x_4}, x_5, x_6\}$ sont ajoutées à la formule tandis que cr_1 et c_6 sont supprimées. Ce processus est répété jusqu'à ce que toutes les clauses de ψ ait été transformées. La figure 2.4 montre les étapes de max-résolution réalisées, avec les clauses de compensation encadrées. Le dernier résolvant produit est $cr_7 = \square$ et on obtient la formule $\Phi_3'' = \{\square, cc_1, \ldots, cc_9\}$ avec $cc_4 = \{\overline{x_4}, x_5, x_6\}$, $cc_5 = \{x_3, \overline{x_4}, \overline{x_5}\}$, $cc_6 = \{\overline{x_3}, x_4, x_5\}$, $cc_7 = \{x_1, \overline{x_2}, \overline{x_3}\}$, $cc_8 = \{x_2, \overline{x_3}\}$ et $cc_9 = \{x_1, x_2\}$.

2.3.4.3 Schéma d'apprentissage

La détection des sous-ensembles inconsistants disjoints, en particulier avec les méthodes basées sur la propagation unitaire (SUP et FL) est coûteuse en temps de calcul. Elle est appliquée à chaque nœud de l'arbre de recherche et les traitements qu'elle réalise sont redondants. En effet, si un sous-ensemble inconsistant ψ est détecté à un nœud donné de l'arbre de recherche, alors ψ sera toujours un sous-ensemble inconsistant dans tous les nœuds de la sous-partie de l'arbre de recherche. Il parait donc logique d'utiliser des méthodes d'apprentissage pour limiter cette redondance et rendre le calcul de la borne inférieure plus incrémental.

La méthode d'apprentissage la plus utilisée est basée sur la transformation par max-résolution, en conservant les changements qu'elle entraîne. Pour limiter l'augmentation de la taille de la formule, seules sont mémorisées les transformations des sous-ensembles inconsistants (ou de parties des sous-ensembles inconsistants) qui correspondent à des motifs prédéfinis. Ces transformations sont souvent décrites comme des règles d'inférences (cf. section 2.2). Nous choisissons ici une description plus générique : l'application de la max-résolution restreinte à certains motifs. On peut distinguer deux types de motifs. Ceux couvrant entièrement un sous-ensemble inconsistant et ceux qui les couvrent partiellement. Les premiers évitent le détection redondante des sous-ensembles inconsistants mémorisés. Leurs transformations génèrent des clauses vides qui peuvent être directement exploitées dans le calcul de la borne inférieure. Les seconds produisent après transformation des clauses résolvantes unitaires. Leur mémorisation permet de réduire

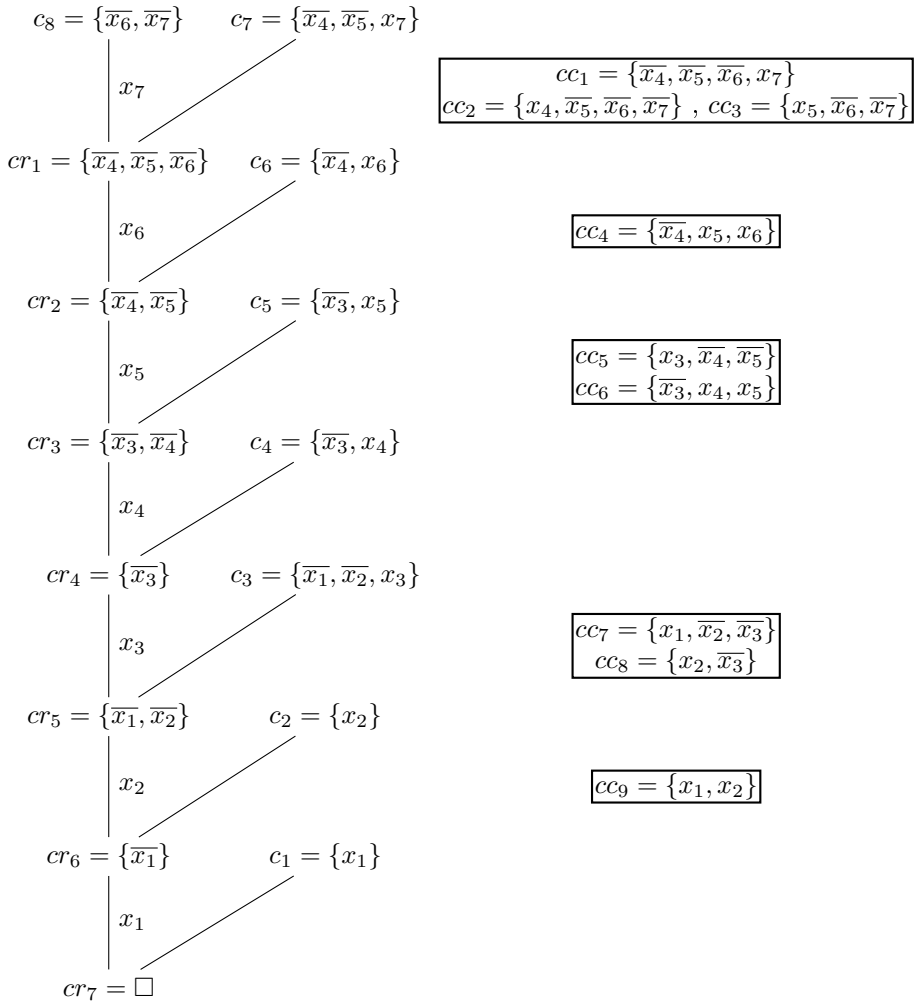


FIGURE 2.4 – Étapes de la max-résolution appliquées sur le sous-ensemble inconsistant ψ de la formule Φ_3 dans l'exemple 10. Les clauses de compensation sont encadrées.

la redondance dans l'application de la propagation unitaire et plus généralement permet d'accroître la capacité de détection des sous-ensembles inconsistants de SUP et FL.

Par exemple, les solveurs WMAXSATZ [LMP07] et AKMAXSAT [Küg10] utilisent principalement les motifs suivants (avec en haut les clauses originales des sous-ensembles inconsistants et en bas les clauses obtenues après transformation) :

$$\frac{\{l_1, l_2\}, \{l_1, \overline{l_2}\}}{\{l_1\}} \tag{2.15}$$

$$\frac{\{l_1, l_2\}, \{\overline{l_2}, l_3\}, \{l_1, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}} \tag{2.16}$$

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_k}, l_{k+1}\}, \{\overline{l_{k+1}}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-1}, \overline{l_k}\}} \tag{2.17}$$

Ces motifs peuvent être combinés. Ainsi, on obtient par 2.15 et 2.17 :

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-2}}, l_{k-1}\}, \{l_{k-1}, l_k\}, \{l_{k-1}, \overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-2}, \overline{l_{k-1}}\}} \tag{2.18}$$

et par 2.16 et 2.17 :

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-3}}, l_{k-2}\}, \{l_{k-2}, l_{k-1}\}, \{l_{k-2}, l_k\}, \{\overline{l_{k-1}}, \overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-3}, \overline{l_{k-2}}\}, \{l_{k-2}, l_{k-1}, l_k\}, \{\overline{l_{k-2}}, \overline{l_{k-1}}, \overline{l_k}\}} \tag{2.19}$$

On peut remarquer que le premier motif (2.15) correspond à la règle d'inférence de la résolution chaînée [LMP07, LHdG08] tandis que les deux suivants (2.16 et 2.17) correspondent à des cas particuliers de résolution cyclique [LHdG08, LMMP09] avec $k = 1$ et $k = 2$ (cf. section 2.2). Il est important de noter que les transformations apprises ne le sont que dans la sous-partie de l'arbre de recherche. Aucun apprentissage vers le haut de l'arbre n'est réalisé.

D'autres manières de limiter l'apprentissage des transformations par max-résolution ont été proposées. Par exemple, le solveur MINIMAXSAT [HLO07, HLO08] ne mémorise une transformation que si tous les résolvants intermédiaires sont de taille strictement inférieure à quatre. Le but est, comme pour les méthodes basées sur les motifs, de limiter l'augmentation de la taille de la formule induite par l'ajout des clauses de compensation. Là encore, les transformations ne sont conservées que dans la sous-partie de l'arbre de recherche.

Enfin, on peut noter que Darras *et al.* ont proposé [DDDL07] de mémoriser certains des sous-ensembles inconsistants supprimés de la formule (qui ne correspondent donc pas aux critères d'apprentissage classiques). Ils ne considèrent cependant que les sous-ensembles inconsistants de petite taille et les transformations ne sont conservées dans la sous-partie de l'arbre de recherche que tant qu'aucune des variables apparaissant dans leurs clauses n'est affectée.

2.3.5 Gestion des clauses dures

La solution d'une instance Max-SAT partiel (valué ou non-valué) est une interprétation satisfaisant toutes les clauses de la partie dure de la formule et minimisant le

nombre de clauses souples falsifiées (ou la somme des poids des clauses souples falsifiées dans le cas d'une instance valuée). L'efficacité des solveurs séparation et évaluation sur ces instances dépend donc de leur capacité à parcourir rapidement les solutions de la partie dure des instances, en particulier lorsque celle-ci est difficile.

Par défaut, un algorithme séparation et évaluation tel que celui présenté dans la section 2.3.1 fonctionne sur la partie dure des instances comme un algorithme DPLL pour SAT (cf. section 1.3.1) : il explore l'arbre de recherche en utilisant la propagation unitaire (réelle) pour étendre l'affectation courante. Quand un conflit est détecté sur la partie dure, il réalise un simple retour-arrière. On peut noter que les solveurs de type séparation et évaluation sont capables de détecter les formules dont la partie dure n'est pas satisfiable : la valeur de la borne supérieure après le parcours complet de l'arbre de recherche est alors supérieure à la somme des poids des clauses souples.

Ce comportement peut être amélioré en incorporant les techniques récentes utilisées dans les solveurs SAT modernes. Parmi les méthodes qui pourraient être appliquées dans un solveur de type séparation et évaluation pour Max-SAT, on peut citer l'apprentissage de clauses, les retours arrières non-chronologiques ou les heuristiques de branchement basées sur les apparitions des variables dans les conflits. Ces méthodes, qui sont utilisées par les solveurs SAT de type *conflict driven clause learning* (CDCL) [ES03, MSS99], se sont avérées efficaces pour résoudre les instances SAT *crafted* et *industrial*. L'implémentation de ces techniques dans un solveur de type séparation et évaluation est cependant complexe puisqu'elle nécessite de faire cohabiter les techniques propres à Max-SAT et celles propres à SAT. À notre connaissance, aucun des solveurs de type séparation et évaluation pour Max-SAT n'inclut ces méthodes (à l'exception de MiniMaxSat [HLO07, HLO08], qui est basé sur le code source du solveur CDCL MiniSat [ES03, SE08]).

2.3.6 Pré-traitement par un algorithme incomplet

L'efficacité des solveurs de type séparation et évaluation dépend de la qualité des bornes supérieure et inférieure. Plus la borne supérieure est proche de la valeur de l'optimum tôt dans la recherche, plus les solveurs pourront effectuer des coupes dans l'arbre de recherche et ainsi limiter le nombre de nœuds explorés. Or, comme nous l'avons vu dans la section 2.3.2, les heuristiques de branchement privilégient les variables permettant d'obtenir un arbre de recherche équilibré, avec des valeurs élevées de la borne inférieure dans le haut de l'arbre de recherche. Cela a pour effet de retarder la découverte d'une borne inférieure de bonne qualité.

Pour remédier à cette situation et permettre aux algorithmes de type séparation et évaluation de commencer la recherche avec une borne supérieure de bonne qualité, Borchers et Furman ont proposé [BF98] d'appliquer un solveur incomplet (typiquement un algorithme de recherche locale) pour obtenir une bonne première valeur de la borne supérieure.

2.4 Recherche Locale

Le problème Max-SAT classique (non valué, non partiel) consiste à trouver un interprétation maximisant le nombre de clauses satisfaites d'une formule CNF en entrée. Par conséquent, la fonction objectif (ou fonction d'évaluation) utilisée par les solveurs de

recherche locale pour SAT est adaptée à la résolution de Max-SAT. Les algorithmes de recherche locale pour SAT peuvent donc être utilisés pour résoudre le problème Max-SAT classique moyennant des modifications mineures.

L'algorithme 2.6 montre un implémentation classique de la recherche locale pour Max-SAT. On voit que contrairement à la recherche locale pour SAT, la meilleure valeur trouvée de la fonction objectif (notée OPT) est mémorisée ainsi que l'interprétation correspondante (notée I_{OPT}). OPT et I_{OPT} sont d'abord initialisés à la pire valeur possible : la somme des poids des clauses souples de la formule (lignes 2-3). Puis, pour chaque interprétation I considérée, OPT est mis à jour si I falsifie moins de clauses que la meilleure interprétation trouvée jusqu'à présent. Lorsque toute les relances ont été effectuées, le couple (OPT, I_{OPT}) est renvoyé. Le reste de l'algorithme est similaire aux méthodes de recherche locale pour SAT (cf. algorithme 1.3).

Algorithme 2.6 : Schéma général d'un algorithme de recherche locale pour Max-SAT.

Data : Une formule CNF Φ.
Result : Un couple (OPT, I_{OPT}) avec I_{OPT} la meilleure solution atteinte par l'algorithme et OPT la somme des poids des clauses falsifiées par I_{OPT} ou INCONNU si aucune solution de la partie dure n'a été trouvée.

```
1  begin
2      OPT ← ∑_{c∈Φ} poids(c);
3      I_OPT ← ∅;
4      for i ← 1 to maxRelances do
5          I ←interprétation_complète_aléatoire();
6          for j ← 1 to maxFlips do
7              if ∑_{c∈Φ, c|I=□} poids(c) < OPT then
8                  OPT ← ∑_{c∈Φ, c|I=□} poids(c);
9                  I_OPT ← I;
10             else
11                 I ←mouvement(Φ,I);
12     if OPT ≤ ∑_{c∈Φ_S} poids(c) then return (OPT, I_OPT) ;
13     else return INCONNU ;
```

2.4.1 Recherche locale pour Max-SAT

Historique La première implémentation d'un algorithme de recherche locale spécifiquement pour Max-SAT est due à Hansen et Jaumard [HJ90], qui ont adapté une méthode de recuit simulé et une méthode de recherche locale avec tabou. Depuis lors, plusieurs adaptations directes ou des combinaisons de solveurs SAT pour Max-SAT ont été réalisées. Parmi elles, on peut citer l'application directe de WalkSAT à Max-SAT [SKC94]. On peut citer également la combinaison de deux types de solveurs incomplets, *Iterated Local Search* [LMS02] et *Robust Tabu Search* [Tai91], réalisée par Smyth *et al.* [SHS03] pour former un nouvel algorithme nommé IRoTS (pour *Iterated Robust Tabu Search*). Plus récemment, des algorithmes basés sur le *configuration checking* [CSS11] ont été adaptés à Max-SAT [LCW+14]. Des méthodes plus complexes ont également été mises en œuvre

pour résoudre Max-SAT par la recherche locale. Nous passons en revue quelques unes de ces méthodes dans la suite de cette section.

Les méthodes de recherche locale considèrent habituellement le voisinage d'une interprétation comme l'ensemble des interprétations accessibles en flippant une variable. Yagiura et Ibaraki [YI98, YI01] ont proposé de considérer l'ensemble des voisinages accessibles en flippant deux ou trois variables simultanément. L'objectif est de permettre à la recherche locale une meilleure anticipation de l'impact de ses mouvements dans l'espace de recherche. Les auteurs redéfinissent le mode de calcul de l'impact d'un mouvement dans ce nouveau contexte et introduisent des techniques pour limiter le nombre de voisins considérés. Ils implémentent cette méthode dans plusieurs algorithmes de recherche locale et les testent sur des instances SAT et Max-SAT. Les résultats obtenus, plutôt décevant sur les instances SAT considérées dans l'évaluation, semblent prometteurs sur les instances Max-SAT et en particulier sur les instances structurées.

Plusieurs algorithmes basés sur l'estimation probabiliste des *backbones*[2] ont été proposées [TS02, ZRL03]. Le principe est de calculer pour chaque littéral sa probabilité d'appartenir au *backbone*. Ces probabilités sont mises à jour à chaque nouvel optimum local, en incrémentant les probabilités associées aux littéraux de l'interprétation courante. Les probabilités servent ensuite à guider la recherche locale dans les différents choix qu'elle effectue (choix de la première interprétation aléatoire, choix de la clause falsifié, choix des variables à flipper, etc.).

Tompkins et Hoos [TH03] ont adapté l'algorithme de recherche locale dynamique SAPS à Max-SAT. Ils font remarquer que des valeurs de facteur d'incrément des pénalités trop élevées peuvent empêcher la détection d'optima locaux (et donc de l'optimum) sur les instances fortement contraintes.

Lardeux *et al.* [LSH05] ont proposé un solveur hybride en introduisant une troisième valeur *unset* dans la recherche locale pour faciliter l'intégration avec les solveurs complets classiques. Un solveur de type séparation et évaluation est appliqué à intervalle régulier pour compléter l'interprétation partielle donnée aux variables par la recherche locale (ie. donner une valeur *vrai* ou $faux$ aux variables *unset*).

Plus récemment, Lü et Hao [LH12] ont introduit un solveur nommé ALMS (pour *Adaptive Memory based Local Search*) incluant une série de techniques destinées à améliorer la diversification et à s'échapper efficacement des optima locaux. Ce solveur inclut (entre autres) : un système de tabou dynamique avec aspiration, un système de pénalité sur les clauses pour guider la diversification, l'adaptation dynamique des probabilités de diversification [Hoo02] et un opérateur de perturbation permettant de se déplacer rapidement vers d'autres portions de l'espace de recherche lorsqu'un optimum local est atteint.

Spécificités du problème Max-SAT Les différentes adaptations faites des méthodes de recherche locale à Max-SAT et les études, théoriques [PBTN12] ou empiriques [ZRL03, CLTS14], qui les ont accompagnées ont permis de dégager une caractéristique principale propre au problème Max-SAT.

Les instances Max-SAT sont généralement non satisfiables et dans de nombreux cas elles contiennent de nombreux sous-ensembles inconsistants. Sur ces instances, le nombre

2. Le *backbone* d'une instance SAT est l'ensemble des littéraux satisfaits par tous les modèles de l'instance [MZK+99].

de clauses falsifiées par chaque interprétation est élevé et on dit qu'elles sont fortement contraintes. Il a été observé [ZRL03, CLTS14] que les méthodes de type WalkSAT (c'est-à-dire qui choisissent la variable à flipper dans une clause falsifiée choisie aléatoirement) sont généralement moins efficaces sur les instances fortement contraintes que les méthodes de type GSAT (qui choisissent la variable à flipper parmi l'ensemble des variables de la formule). Ceci peut s'expliquer par le fait que sur les instances fortement contraintes de nombreuses clauses sont falsifiées. Par conséquent, l'heuristique de choix de la variable à flipper utilisée par WalkSAT peut provoquer un aléa trop important dans le choix des mouvements et ainsi réduire l'efficacité de l'intensification sur les zones prometteuses de l'espace de recherche.

2.4.2 Recherche locale pour Max-SAT valué

L'adaptation de la recherche locale à Max-SAT valué peut être faite naturellement, en incluant le poids des clauses dans la fonction d'évaluation utilisée. On évaluera les interprétations en fonction de la somme des poids des clauses qu'elles falsifient plutôt que sur leur nombre. La plupart des algorithmes de recherche locale pour SAT (et donc pour Max-SAT classique) peuvent être utilisés pour résoudre les instances Max-SAT valué avec cette nouvelle fonction d'évaluation.

Un problème se pose cependant pour les algorithmes de recherche locale dynamique, qui attribuent tout au long de la recherche des poids aux clauses selon leur difficulté (nous parlerons de pénalités pour éviter toute confusion avec les poids des clauses valuées). Il faut faire cohabiter dans la fonction objectif les deux systèmes de valuation des clauses. Plusieurs solutions ont été retenues par le passé. Dans un algorithme pour Max-SAT basé sur la modélisation discrète de Lagrange (structurellement proche de la recherche locale dynamique), Wu et Wah [WW99, WW00] utilisent une fonction d'évaluation basée sur la somme du poids et de la pénalité des clauses. L'algorithme GLSSAT [MT00] utilise le poids des clauses dans la mise à jour des pénalités, mais pas directement dans la fonction objectif. A notre connaissance, il n'existe pas d'étude comparative sur le sujet.

2.4.3 Recherche locale pour Max-SAT partiel

Le problème Max-SAT partiel pose un autre défi. La solution optimale d'une instance Max-SAT partiel doit satisfaire la partie dure de la formule tout en minimisant le nombre (ou la somme des poids) des clauses souples falsifiées. Cependant, l'optimum n'est pas nécessairement atteignable pour toute interprétation satisfaisant la partie dure du problème. La recherche locale doit donc énumérer les modèles de la partie dure pour maximiser ses chances de pouvoir trouver une solution optimale. Mais d'un autre coté, elle doit également consacrer suffisamment de temps à l'optimisation de la partie souple pour chaque modèle de la partie dure trouvé. Plusieurs composants peuvent avoir un impact sur cet équilibre. La fonction d'évaluation utilisée doit privilégier la satisfaction des clauses dures mais pas trop pour limiter le "pouvoir d'attraction" des modèles de la partie dure. L'heuristique de choix des mouvements et les méthodes de diversification (marche aléatoire, tabou, etc.) peuvent compenser dans une certaine mesure le rôle de la fonction d'évaluation.

On peut distinguer deux approches dans la manière de concilier clauses dures et clauses souples. La première consiste à utiliser une seule fonction objectif intégrant les

deux types de clauses et la seconde approche consiste à utiliser deux fonctions objectif distinctes. Nous présentons dans la suite de cette section les implémentations existantes pour ces deux approches.

Fonction objectif unique La manière la plus simple de concilier clauses dures et clauses souples est de les intégrer dans une fonction objectif unique, en pondérant les clauses dures pour privilégier la satisfaction de la partie dure de l'instance. Une telle fonction objectif peut s'écrire :

$$eval(\Phi, I) = \delta * |\{c_j \in \Phi_D,\ c_j|_I = \square\}| + \sum_{c_j \in \Phi_S,\ c_j|_I = \square} poids(c_j)$$

avec δ le poids attribué aux clauses dures, Φ_D la partie dure de la formule et Φ_S la partie souple. Les premières implémentations de recherche locale pour Max-SAT partiel utilisant une fonction objectif unique [JKS95] choisissaient une valeur de δ supérieure à la somme des poids des clauses souples. Cependant, le pouvoir d'attraction vers les optima locaux des modèles de la partie dure ainsi obtenue est trop important, ce qui peut nuire à l'optimisation de la partie souple. Pour remédier à ce problème, Cha *et al.* [CIKM97] ont proposé d'adapter la valeur de δ au cours de la recherche en la fixant à la somme des poids des clauses souples falsifiées. Une méthode similaire est utilisée par [TBSP02] dans leur algorithme à deux niveaux. δ est incrémenté tant que la partie dure de l'instance n'est pas satisfaite, et décrémenté si un modèle de la partie dure est trouvé jusqu'à atteindre la somme des poids des clauses souples falsifiées.

Fonctions objectifs distinctes L'autre approche consiste à utiliser deux fonctions objectifs, une pour les clauses dures $eval_D$ et une pour les clauses souples $eval_S$. L'équilibre entre le temps passé à satisfaire la partie dure et celui passé à optimiser la partie souple est alors entièrement laissé à la discrétion de l'heuristique de mouvement.

$$eval_D(I) = |\{c_j \in \Phi_D,\ c_j|_I = \square\}|$$

$$eval_S(I) = \sum_{c_j \in \Phi_S,\ c_j|_I = \square} poids(c_j)$$

Cette méthode est utilisée notamment dans le solveur Dist [CLTS14], qui choisit en priorité un mouvement améliorant $eval_D$. Si aucun n'existe, le solveur choisit un mouvement qui ne dégrade pas $eval_D$ et qui améliore $eval_S$. S'il n'existe pas de tel mouvement, il choisit aléatoirement une clause falsifiée (dure de préférence) et flippe la variable qui améliore le plus $eval_S$.

2.5 Autres méthodes de résolution

Outre les algorithmes de type séparation et évaluation et ceux de recherche locale, d'autres méthodes ont été utilisées pour résoudre le problème Max-SAT. Nous présentons brièvement dans cette section les principales approches existantes et décrivons pour chacune certains des travaux les plus récents.

2.5.1 Application itérative de solveurs SAT

Le problème Max-SAT peut être résolu par des appels successifs à un solveur SAT complet. Un des principaux avantages de cette approche est qu'elle bénéficie directement des travaux et avancés réalisés sur la résolution du problème SAT. En particulier, les bonnes performances des solveurs SAT de type CDCL rendent ces méthodes particulièrement efficaces sur les instances fortement structurées (e.g. industrielles). Deux approches existent.

2.5.1.1 Transformation en problèmes de décision

Une instance du problème Max-SAT peut être transformée en une série de problèmes de décision pouvant être résolus par un solveur SAT. Pour une formule Φ et un entier k, on peut déterminer grâce à un solveur SAT s'il existe une interprétation falsifiant k clauses de Φ. On note Φ'_k la formule correspondante à ce problème de décision. Si Φ'_k est satisfiable, alors $k+1$ est une borne supérieure (BS) de l'optimum de l'instance. Si elle n'est pas satisfiable, alors k est une borne inférieure (BI) de l'optimum.

Ce principe est utilisé pour résoudre le problème Max-SAT, en parcourant l'intervalle des valeurs d'optimum possibles pour faire converger les bornes supérieures et inférieures. Plusieurs types de parcours de cet intervalle ont été proposés.

- Parcours linéaire [FM06, BP10, KZFH12], qui consiste à faire varier k de 0 à $\sum_{c_i \in \Phi} poids(c_i)$ jusqu'à trouver une instance Φ'_k satisfiable ou inversement de $\sum_{c_i \in \Phi} poids(c_i)$ à 0 jusqu'à trouver une instance non-satisfiable.
- Parcours binaire [FM06, KZFH12], qui commence par initialiser BI à -1 et BS à $\sum_{c_i \in \Phi} poids(c_i) + 1$. Puis, itérativement, k est fixé à la valeur médiane de l'intervalle $]BI, BS[$ et les valeurs de BI et BS sont mises à jour selon que Φ'_k est satisfiable ou non. Ce processus est répété jusqu'à ce que BI et BS convergent.
- Parcours hybride [KZFH12], qui alterne entre les parcours linéaire et binaire.

La conversion de la formule originale Φ en un problème de décision Φ'_k se fait en ajoutant à chaque clause souple c_i de Φ une variable de relaxation b_i, puis en ajoutant à la formule une contrainte de cardinalité autorisant au maximum k variables de relaxation à être satisfaites. La manière de convertir la contrainte linéaire en forme clausale a une incidence sur les propriétés de la formule transformée, comme le nombre de ses variables et de ses clauses. Les lecteurs intéressés peuvent se référer au chapitre 22 de *Handbook of Satisfiability* [BHvMW09] qui recense les différents types de codage existants.

Parmi les algorithmes et solveurs basés sur cette approche, on peut citer sat4j [BP10] ou QMAXSAT [KZFH12].

2.5.1.2 Guidage par les sous-ensembles inconsistants

La seconde approche se base sur la capacité des solveurs SAT complets de fournir un sous-ensemble inconsistant lorsque la formule en entrée est non-satisfiable. Seules les clauses appartenant à ces sous-ensembles inconsistants ont besoin d'être relaxées. Cela permet d'ajouter moins de variables de relaxation à la formule et donc moins de clauses lors de l'encodage des contraintes de cardinalité en forme clausale.

Cette approche a été introduite par Fu et Malik [FM06], puis améliorée à de nombreuses reprises [MM08, MSP08, MMSP09, ABL09b]. Ces améliorations ont principalement porté sur la manière de relaxer les clauses apparaissant dans les sous-ensembles

inconsistants, par exemple en réduisant le nombre de variables de relaxation utilisées ou en utilisant d'autres contraintes de cardinalités [MM08]. L'exploitation des relations entre les sous-ensembles inconsistant détectés a aussi permis d'améliorer l'efficacité de cette approche [ABL09b]. Marques-Silva et Planes [MSP08] et plus récemment Heras *et al.* [HMMS11] ont également proposé des solveurs hybrides qui cherchent à combiner les avantages de la transformation en problème de décision et du guidage par les cœurs inconsistants.

Les évolutions les plus récentes ont vu l'ajout de techniques complexes, comme l'utilisation de méthodes issues de la programmation linéaire en nombres entiers pour gérer les contraintes de cardinalité [DB11, DB13a], la transformation des sous-ensembles inconsistants par max-resolution [HMS11, NB14], la prise en compte des symétries [ABGL12] ou le partitionnement des clauses souples [MML12].

De nombreux algorithmes basés sur cette approche ont été proposés au cours des dix dernières années. Parmi eux, on peut citer WMSU1/WPM1 [FM06, ABL09b, MMSP09], MSU [MM08, MP07, MSP08], PM/WPM [ABL09a, ABL09b, ABGL12, ABL10, ABL13], WBO [MML10, MML13], PAR [MML12], MAXHS [DB11], BIN-C [HMMS11, MHM12], EVA [NB14].

2.5.2 Reformulations et relaxations en d'autres problèmes d'optimisation

Nous avons vu que les instances d'un nombre important de problèmes d'optimisation pouvaient être transformées en instances Max-SAT. À l'inverse, les méthodes de résolution issues d'autres problèmes d'optimisation peuvent être utilisées pour résoudre des instances Max-SAT, soit en les appliquant directement après transformation des instances soit en les intégrant dans une méthode de résolution pour Max-SAT. Nous distinguerons deux types de transformation : les *reformulations* qui préservent l'équivalence des instances transformées et les *relaxations* qui peuvent occasionner la perte de certaines contraintes lors de la transformation. Dans ce second cas, l'algorithme appliqué sur la formule transformée ne pourra garantir l'optimalité de la solution qu'il renvoie.

Programmation linéaire en nombres entiers (*integer linear programming*, ILP) La programmation linéaire en nombres entiers (*integer linear programming*, ILP) a été utilisée de diverses manières pour résoudre Max-SAT. La relaxation en ILP a été utilisée pour fixer les valeurs des variables dans un algorithme *branch and cut* [JMB97] ou pour calculer l'estimation de la borne inférieure dans un algorithme *branch and bound*, soit par relaxation directe de la formule [XZ05] soit en estimant la taille du plus petit ensemble couvrant des sous-ensembles inconsistants [DCB10]. Un solveur ILP est également utilisé dans le solveur itératif SAT MAXHS [DB11, DB13a, DB13b]. Plutôt que de relaxer les clauses des sous-ensembles inconsistants détectés par le solveur SAT et d'ajouter une contrainte de cardinalité comme décrit dans la section 2.5.1.2, MAXHS détermine grâce à un solveur ILP un ensemble couvrant des sous-ensembles inconsistants puis il supprime les clauses de cet ensemble couvrant de la formule. Enfin, des solveurs comme SCIP-MAXSAT ou ILP-2013 [AG13] transforment directement les instances Max-SAT puis les résolvent en appelant un solveur ILP (respectivement SCIP [Ach09] et IBM-CPLEX).

Programmation semi-définie (*semi-definite programming*, SDP) La relaxation des instances Max-SAT en programmation semi-définie à été majoritairement utilisée comme méthode d'approximation [GW94a, GW95, KZ97] (cf. section 2.5.3). Une comparaison de la qualité des relaxation en ILP et SDP a été réalisé par Gomes *et al.* [GvHL06]. Les lecteurs intéressés par les applications de la programmation semi-définies pour SAT et Max-SAT peuvent se référer au papier de Anjos [Anj06].

Answer set programming (ASP) Les instances Max-SAT peuvent également être reformulées en instances ASP puis résolues par un solveur dédié comme le portfolio CLASP [GKK+11, GKKS12, POT].

Problème de satisfaction de contraintes (*Constraint satisfaction problem*, CSP Enfin, de Givry *et al.* ont proposé [dGLMS03] une méthode basée sur la reformulation en CSP. Les instances sont converties en un réseau de contraintes, puis résolues par le solveur CSP TOOLBAR [LS03, dGHZL05]

2.5.3 Méthodes d'approximation

La recherche locale permet, sur certaines classes d'instances, de trouver rapidement une solution proche de l'optimalité. Cependant, elle n'offre aucune garantie sur la qualité de la solution renvoyée. À l'inverse, les algorithmes d'approximation fournissent une borne inférieure à la qualité de la solution qu'ils renvoient, exprimée sous la forme d'un ratio de la valeur de l'optimum. Ainsi, pour une constante α donnée, le nombre de clauses falsifiées par la solution d'un algorithme α-approché pour Max-SAT est toujours inférieur à $1/\alpha$ fois la valeur de l'optimum, et ce quelque soit l'instance en entrée.

Les premières méthodes approximatives pour Max-SAT étaient basées sur des algorithmes gloutons [Joh73, Joh74] garantissant des performances de 1/2. Ce seuil a été amélioré par la suite, par Yannakakis [Yan92, Yan94] et Goemans et Williamson [GW94b] qui ont utilisé respectivement des techniques d'optimisation des flots dans les graphes et la relaxation en programmation linéaire pour atteindre une garantie de performance de 3/4. Håstad [Hås97, Hås01] a prouvé que, à moins que $P = NP$, la meilleure garantie atteignable par un algorithme d'approximation pour Max-SAT était 7/8. Cette limite a été atteinte par Karloff et Zwick [KZ97] qui ont proposé un algorithme basé sur la relaxation en programmation semi-définie, étendant ainsi les travaux réalisés par Goemans et Williamson [GW94a, GW95].

2.6 Évaluation expérimentale des solveurs Max-SAT complets

Nous présentons dans cette section la méthodologie et les benchmarks utilisés lors des *Max-SAT evaluations* pour comparer les solveurs complets pour Max-SAT, puis nous introduisons le protocole que nous avons utilisé durant cette thèse pour évaluer expérimentalement les méthodes de résolution pour Max-SAT.

2.6.1 Protocole et instances de la *Max-SAT evaluation 2014*

Les solveurs complets testés durant la *Max-SAT evaluation 2014* disposent de 1800 secondes pour résoudre les instances et la mémoire vive qui leur est allouée est limitée à 3 Go. Le premier critère permettant de comparer l'efficacité des solveurs complets est bien évidemment le nombre d'instances qu'ils résolvent. En cas d'égalité sur ce premier critère, le temps moyen de résolution peut donner un estimation plus précise des performances respectives des solveurs.

Durant cette évaluation, les tests ont été réalisés sur des machines équipées de processeurs Intel Xeon cadencés à 2 GHz, de 3.5 Go de mémoire vive et fonctionnant sous un système d'exploitation GNU/Linux.

Les instances utilisées[3] se divisent en trois catégories correspondant aux variantes du problème Max-SAT : Max-SAT classique (noté ms dans les tableaux de résultats), Max-SAT partiel (noté pms) et Max-SAT partiel valué (noté wpms). Chacune de ces catégories comprend elle-même trois subdivisions :

— les instances *random* qui, comme leur nom l'indique, sont générées aléatoirement,
— les instances *crafted* qui résultent généralement de la transformation d'instances d'autres problèmes académiques en instances Max-SAT,
— les instances *industrial* qui sont issues de problèmes réels.

Nous montrons dans la table 2.1 les statistiques détaillées de ces instances. Le nombre de variables (colonne V) varie de 50 à presque 200000. On peut noter cependant que la moyenne de 1719 variables par instance est tirée vers le haut par quelques classes d'instances. Le nombre de variables médian est de 150. On observe le même phénomène pour le nombre de clauses (colonne C) par instance, qui varient de 461 à plus de 900000, avec une moyenne de 17230 et une médiane de 1500. Le ratio clauses/variable (colonne r) est généralement élevé (10 en moyenne, allant de 2,1 à 132,2).

Si l'on considère séparément le nombre de clauses et le ratio des clauses dures (colonnes C_D et r_D) et souples (colonnes C_S et r_S), on peut remarquer qu'à quelques exceptions près, les instances Max-SAT partiel (catégories pms et wpms) ont sensiblement plus de clauses dures que de clauses souples. On peut même voir que sur ces instances, le ratio du nombre de clauses souples par variable est rarement supérieur à un. Cela montre que la difficulté de ces instances réside davantage dans l'énumération des solutions de la partie dure que dans l'optimisation de la partie souple.

Les clauses (dures ou souples, colonnes C_D tailles et C_S tailles) comprennent généralement entre un et trois littéraux. On peut remarquer cependant que sur certaines classes d'instances, des clauses peuvent inclure plus de mille littéraux.

Enfin, on peut remarquer une différence entre les poids des clauses souples des instances *crafted* et *random* (colonnes C_S poids, lignes wpms). Les poids des premières peuvent prendre des valeurs élevées (jusqu'à plus de 14 millions) tandis que ceux des secondes ne dépassent jamais 10. Nous discuterons dans le chapitre 8 de l'influence que cela peut avoir sur le comportement des solveurs de type séparation et évaluation.

2.6.2 Protocole et instances utilisés dans ce document

Tous les tests dont les résultats sont présentés dans cette thèse ont été réalisés sur un cluster de 16 machines équipées de processeurs Intel Xeon cadencés à 2.4 GHz, de 24 Go

3. Disponibles à l'adresse `http://www.maxsat.udl.cat/14/benchmarks/`

TABLE 2.1 – Caractéristiques des instances de la *Max-SAT Evaluation 2014*. Les deux premières colonnes montrent les classes d'instances et leurs tailles. Les colonnes V, C et r donnent respectivement les moyennes du : nombre de variables, nombre de clauses et ratio clauses/variable. Les colonnes C_D et r_D (C_S et r_S) donnent les moyennes du nombre de clauses dures (souples) et du ratio clauses dures (souples)/variable. Les six dernières colonnes donnent respectivement les moyennes, minimums et maximums des tailles des clauses dures et souples et du poids des clauses.

Classes d'instances		#	V	C	r	C_D	r_D	C_S	r_S	C_D tailles		C_S tailles		C_S poids	
										moy	[min,max]	moy	[min,max]	moy	[min,max]
ms	crafted	bipartite	100	140	1259	8,99	0	0,00	1259	8,99	-	-	2,0	-	1	-
		maxcut	67	50	1099	21,82	0	0,00	1099	21,82	-	-	2,0	[1,2]	1	-
		set-covering	10	2210	7677	3,47	0	0,00	7677	3,47	-	-	19,3	[1,495]	1	-
	random	highgirth	82	222	1159	5,22	0	0,00	1159	5,22	-	-	3,4	[3,4]	1	-
		max2sat	100	130	1400	10,77	0	0,00	1400	10,77	-	-	2,0	-	1	-
		max3sat	100	75	900	12,00	0	0,00	900	12,00	-	-	3,0	-	1	-
		min2sat	96	180	1800	10,00	0	0,00	1800	10,00	-	-	1,5	[1,2]	1	-
pms	crafted	frb	25	470	21184	45,07	20714	44,07	470	1,00	2,0	-	1,0	-	1	-
		job-shop	3	84607	877437	10,37	877244	10,37	193	0,00	2,8	[1,3]	1,0	-	1	-
		maxclique	158	281	23143	82,30	22862	81,30	281	1,00	2,0	-	1,0	-	1	-
		maxone	140	301	1453	4,82	1152	3,82	301	1,00	3,2	[1,81]	1,0	-	1	-
		min-enc/kbtree	42	280	1261	4,50	364	1,30	897	3,20	2,2	[2,5]	3,0	[1,5]	1	-
		pseudo/miplib	4	224	461	2,06	437	1,95	24	0,11	2,6	[1,7]	1,0	-	1	-
		reversi	44	21534	129243	6,00	129182	6,00	61	0,00	2,5	[1,47]	1,0	-	1	-
		scheduling	5	189125	749732	3,96	748270	3,96	1462	0,01	2,5	[1,265]	1,0	-	1	-
	random	min2sat	60	920	4528	4,92	4066	4,42	462	0,50	2,0	-	2,0	[1,3]	1	-
		min3sat	60	375	4314	11,50	4154	11,08	160	0,43	2,0	-	2,3	[1,5]	1	-
		pmax2sat	60	150	3750	25,00	150	1,00	3600	24,00	2,0	-	2,0	-	1	-
		pmax3sat/hi	30	100	700	7,00	100	1,00	600	6,00	3,0	-	3,0	-	1	-
wpms	crafted	auction	40	176	4967	28,29	4791	27,29	176	1,00	2,0	-	1,0	-	687	[19,2055]
		CSG	10	6859	906625	132,19	906538	132,18	87	0,01	3,0	[1,7]	1,1	[1,6]	1673	[3,10356]
		frb	34	373	16047	43,08	0	0,00	16047	43,08	-	-	2,0	[1,2]	364	[1,761]
		min-enc	74	2147	18423	8,58	17803	8,29	620	0,29	3,5	[1,51]	1,0	-	104715	[1,14156394]
		pseudo/miplib	12	10523	41657	3,96	41033	3,90	624	0,06	2,9	[1,93]	1,0	-	760	[1,11000]
		ramsey	15	117	3310	28,21	0	0,00	3310	28,21	-	-	5,3	[3,6]	504	[1,1000]
		random-net	32	437	1817	4,16	146	0,33	1671	3,83	3,0	[2,4]	2,4	[1,3]	1957	[2,15705]
		set-covering	45	4111	4556	1,11	444	0,11	4111	1,00	262,8	[8,1086]	1,0	-	51	[1,100]
		wmaxcut	48	54	1219	22,46	0	0,00	1219	22,46	-	-	2,0	-	8092	[1,375001]
	random	wmax2sat	120	120	1394	11,62	0	0,00	1394	11,62	-	-	2,0	-	5	[1,10]
		wmax3sat	40	70	850	12,14	0	0,00	850	12,14	-	-	3,0	-	6	[1,10]
		wpmax2sat	90	150	3000	20,00	150	1,00	2850	19,00	2,0	-	2,0	-	5	[1,10]
		wpmax3sat/hi	30	100	700	7,00	100	1,00	600	6,00	3,0	-	3,0	-	5	[1,10]
Ensemble		1776	1719	17230	10,02	15754	9,16	1476	0,86	14,6	[1,1086]	2,0	[1,495]	4661	[1,14156394]

de mémoire vive et fonctionnant sous un système d'exploitation GNU/Linux.

Comme lors des *Max-SAT evaluation*, nous avons fixé le temps maximal de résolution et la mémoire allouée à chaque solveur complet à respectivement de 1800 secondes et 3.5 Go de mémoire vive.

Nous comparerons les performances des solveurs en fonction du nombre d'instances qu'ils résolvent et de leur temps moyen de résolution. Nous utilisons également dans cette thèse des indicateurs permettant de comprendre le fonctionnement interne des solveurs de type séparation et évaluation, comme par exemple le nombre de nœuds explorés dans l'arbre de recherche ou le nombre de sous-ensembles inconsistants détectés à chaque nœud lors de l'évaluation de la borne inférieure. La signification à donner aux variations des valeurs de ces indicateurs peut varier selon les composants étudiés ou les modifications apportées aux solveurs. Nous les préciserons dans chacun des chapitres où ils seront utilisés.

La majorité des tests dont les résultats sont présentés dans la seconde partie de ce document ont été réalisés sur les instances issues de la *Max-SAT evaluation 2014*. Les solveurs étudiés dans la suite de cette thèse (de type séparation et évaluation et de type recherche locale) étant notoirement inefficaces sur les instances *industrial*, nous limiterons les évaluations expérimentales aux instances *random* et *crafted*. Nous reviendrons sur les raisons de cette sélection dans les chapitres 8 et 9.

Dans certains cas, nous utiliserons d'autres instances issues des précédentes *Max-SAT evaluations* ou générées aléatoirement. Nous décrirons ces instances dans les chapitres où elles seront utilisées.

2.7 Conclusion

Nous avons présenté dans ce chapitre le problème Max-SAT et ses variantes, Max-SAT valué et Max-SAT partiel. Nous avons passé en revue les principales règles d'inférence applicables à Max-SAT avant de présenter les principales méthodes de résolution qui peuvent lui être appliquées. Nous avons détaillé le fonctionnement et les principaux composants des solveurs de type séparation et évaluation. Nous avons vu en particulier les méthodes récentes utilisées pour la calcul de la borne inférieure : la détection des sous-ensembles inconsistants par propagation unitaire simulée et leur transformation par la règle de la max-résolution. Nous avons également rappelé les méthodes d'apprentissage utilisées par ces solveurs. Nous avons également discuté de l'applicabilité des méthodes de recherche locale pour SAT au problème Max-SAT et à ses variantes et de l'impact de leur particularités respectives sur le comportement des solveurs de recherche locale. Enfin, nous avons présenté brièvement les autres méthodes de résolution applicables à Max-SAT avant d'introduire la méthodologie expérimentale qui sera utilisée dans la suite de ce document pour l'évaluation des solveurs complets pour Max-SAT.

Deuxième partie

Contributions

Nous avons vu que les solveurs de type séparation et évaluation calculent la borne inférieure en estimant le nombre de sous-ensembles inconsistants disjoints présents dans la formule simplifiée par l'interprétation courante. Ils utilisent des méthodes basées sur la propagation unitaire (la propagation unitaire simulée et la méthode des littéraux contradictoires) pour détecter les sous-ensembles inconsistants puis ils les transforment pour assurer leur caractère disjoint. Une de ces méthodes de transformation, basée sur la règle de la max-résolution, permet de conserver l'équivalence de la formule et peut donc être utilisée comme une méthode d'apprentissage. Cependant, les transformations par max-résolution entraînent l'ajout de clauses de compensation à la formule et par conséquent sa taille peut croître rapidement. De plus, l'impact de ces transformations sur le comportement des solveurs de type séparation et évaluation est encore méconnu. Il a été aussi observé expérimentalement [HLO08] que leur conservation systématique avait un impact négatif sur les performances des solveurs. Pour ces raisons, l'apprentissage n'est appliqué que sur certains sous-ensembles inconsistants et les transformations ne sont conservées que dans la sous-partie de l'arbre de recherche.

Les solveurs de type séparation et évaluation dominent généralement les autres types de solveurs sur les instances faiblement structurées (aléatoires et certaines *crafted*). Cependant, ils restent perfectibles. Les traitements qu'ils appliquent lors du calcul de la borne inférieure sont coûteux en temps de calcul et très redondants puisque les mêmes sous-ensembles inconsistants peuvent être détectés et transformés à de nombreux nœuds de l'arbre de recherche. De plus, ces solveurs sont notoirement inefficaces sur les instances structurées (industrielles et certaines *crafted*). L'absence de mécanisme permettant de faire remonter vers le haut de l'arbre de recherche les informations collectées durant la résolution fait qu'ils sont incapables d'exploiter les propriétés structurelles des instances. L'apprentissage semble un moyen pertinent de remédier à ces faiblesses et par là même d'améliorer les performances des solveurs de type séparation et évaluation.

Pour répondre à ces problémes, nous avons étudié pendant cette thèse les méthodes permettant de réaliser un apprentissage : l'utilisation combinée des règles de la propagation unitaire et de la max-résolution. L'objectif est à la fois d'améliorer le calcul de la borne inférieure (soit en améliorant sa qualité soit en réduisant le temps de calcul qui y est consacré) et de mieux comprendre le fonctionnement et l'impact de ces méthodes. Cela pourrait permettre, à terme, de maîtriser ou limiter les inconvénients des transformations par max-résolution et donc d'élaborer des méthodes d'apprentissage plus globales. Les contributions présentées dans la suite de cette partie sont les suivantes.

Dans le chapitre 3 nous présentons un nouveau schéma d'application de la propagation unitaire qui considère toutes les sources de propagation des variables plutôt qu'uniquement la première comme c'est le cas habituellement. Nous montrons que ce schéma est mieux adapté au contexte de Max-SAT puisqu'il permet de réduire la redondance dans les étapes de propagation unitaire simulée. Nous proposons également une heuristique exploitant les informations disponibles grâce à ce schéma pour influer sur les caractéristiques des sous-ensembles inconsistants détectés lors du calcul de la borne inférieure.

Nous étudions dans le chapitre 4 la relation entre l'ordre d'application des étapes de max-résolution et le nombre et la taille des clauses de compensation produites. Nous montrons que, dans certains cas, appliquer d'abord les étapes de max-résolution produisant les plus petites clauses résolvantes permet de réduire le nombre et la taille des clauses de compensation produites. Nous proposons une heuristique exploitant cette pro-

priété. L'étude expérimentale que nous avons menée montre que cette heuristique réduit significativement le nombre et la taille des clauses de compensation.

Nous proposons dans le chapitre 5 une nouvelle méthode de transformation des sous-ensembles inconsistants détectés durant le calcul de la borne inférieure. Elle consiste à appliquer la max-résolution sur les clauses des sous-ensembles inconsistants localement à chaque nœud de l'arbre de recherche, c'est-à-dire sans appliquer d'apprentissage. Cela permet d'améliorer significativement la qualité de l'estimation de la borne inférieure et donc de réduire le nombre de nœuds explorés dans l'arbre de recherche. Le schéma de transformation des sous-ensembles inconsistants ainsi obtenu est plus homogène : tous les sous-ensembles inconsistants sont transformés par max-résolution, et seule la portée de ces changements (locaux ou conservés dans la sous-partie de l'arbre de recherche) varie. L'évaluation expérimentale que nous avons conduit montre que sur la plupart des classes d'instances considérées les performances des solveurs sont significativement améliorés.

Dans le chapitre 6, nous avons cherché à étendre l'apprentissage réalisé par les solveurs de type séparation et évaluation en incorporant de nouveaux ensembles de motifs dans leur schéma d'apprentissage. Nous évaluons expérimentalement l'impact de ces nouveaux ensembles de motifs. Les résultats obtenus montrent que certains de ces ensembles permettent d'accroître la part d'apprentissage réalisé par les solveurs et donc d'améliorer leurs performances. Ils montrent également que certaines transformations par max-résolution ont un impact négatif sur les performances des solveurs, sans que l'augmentation de la taille de la formule permette d'expliquer ce comportement.

Nous étudions ce phénomène dans le chapitre 7. Nous montrons que les transformations par max-résolution ont un impact sur l'efficacité du mécanisme de la propagation unitaire. Nous introduisons la notion d'UP-résilience des transformations pour caractériser les transformations n'ayant pas d'impact négatif sur la propagation unitaire et plus généralement pour quantifier cet impact. Nous discutons des propriétés associées à l'UP-résilience et montrons que les transformations correspondant aux schémas d'apprentissage existants sont UP-résilients. Cela contribue à expliquer d'un point de vue théorique l'efficacité de ces schémas, qui avait été démontrée uniquement de manière empirique jusqu'à présent. Enfin, nous étudions le comportement de l'ordre d'application des étapes de max-résolution et des schémas d'apprentissages existants au regard de l'UP-résilience. Les résultats obtenus confirment la pertinence de la notion d'UP-résilience et ouvrent plusieurs perspectives d'amélioration.

Le chapitre 8 clos nos contributions sur les solveurs de type séparation et évaluation. Nous y donnons une vue d'ensemble du solveur AHMAXSAT que nous avons développé durant cette thèse pour tester empiriquement nos contributions. Puis nous le comparons expérimentalement à certains des solveurs complets de l'état de l'art les plus performants. Les résultats obtenus montrent qu'AHMAXSAT est concurrentiel voire dépasse les autres solveurs sur plusieurs catégories d'instances.

Enfin, dans le chapitre 9 nous présentons une exploitation de la technique de détection et de transformation des sous-ensembles inconsistants dans un algorithme de recherche locale. Nous discutons des obstacles posés par l'intégration des règles d'inférence de la propagation unitaire et de la max-résolution et nous détaillons les solutions que nous avons apportées. Les résultats expérimentaux obtenus montrent une nette amélioration des performances, même si elle ne permet pas encore de concurrencer les méthodes de recherche locale les plus récentes pour Max-SAT.

Chapitre 3

Nouveau schéma d'application de la propagation unitaire

La méthode de la propagation unitaire (UP), qui consiste à satisfaire itérativement les littéraux apparaissant dans des clauses unitaires (cf. section 1.2.3.2), est un des composants principaux des solveurs de type séparation et évaluation récents. Elle est utilisée à chaque nœud de l'arbre de recherche pour détecter les sous-ensembles inconsistants dans le calcul de la borne inférieure (propagation unitaire simulée et littéraux contradictoires, cf. section 2.3.4) et dans une moindre mesure pour étendre l'interprétation courante (propagation unitaire réelle, cf. section 2.3.3). Par conséquent, le temps passé à appliquer la propagation unitaire représente une part importante du temps d'exécution total des solveurs de type séparation et évaluation.

A notre connaissance, toutes les implémentations existantes de la propagation unitaire utilisent un schéma basé sur la première source de propagation des variables (*first propagation source scheme*, FPS). Dans ce schéma, seule la première clause unitaire causant la propagation des variables est mémorisée (les suivantes sont simplement ignorées). Ce comportement est satisfaisant dans le cadre des solveurs SAT de type DPLL : les propagations sont défaites dans l'ordre chronologique inverse (lors des retours arrières). Lorsqu'une propagation est défaite, toutes ses sources de propagation ignorées ne sont plus unitaires puisque les affectations les réduisant ont été défaites préalablement. La situation est différente dans le cadre des solveurs de type séparation et évaluation pour Max-SAT. Le traitement appliqué aux clauses des sous-ensembles inconsistants (que ce soit la suppression ou la transformation par max-résolution, cf. section 2.3.4) supprime

des clauses de la formule. Par conséquent, une propagation peut être défaite lorsque sa source de propagation est supprimée. Dans une telle situation, toutes les affectations faites après la propagation de la variable doivent être défaites également pour s'assurer que les sources de propagations ignorées précédemment sont maintenant considérées. Cela peut conduire à défaire et refaire inutilement de nombreuses propagations.

Pour corriger ce comportement, nous avons proposé un nouveau schéma d'application de la propagation unitaire dans lequel toutes les sources de propagation des variables sont considérées plutôt que seulement la première. Nous appelerons ce schéma MPS (*Multiple Propagation Source scheme*). À notre connaissance, ce type de schéma d'application de la propagation unitaire n'a été étudié que d'un point de vue théorique jusqu'à présent [VG11]. On peut noter cependant qu'Audemard *et al.* ont utilisé de manière limitée les autres sources de propagation (qu'ils désignent par arcs inverses) lors de l'analyse des conflits pour améliorer le niveau des retour-arrières non-chronologiques (*backjump*) dans un solveur CDCL [ABH+08]. Le schéma de propagation que nous présentons ici permet de défaire les étapes de propagation dans un ordre quelconque, sans être contraint par l'ordre dans lequel elles ont été réalisées. Cela permet de réduire le nombre de propagations défaites et refaites inutilement. À notre connaissance, un tel schéma n'a jamais été implémenté auparavant, que se soit pour SAT ou pour Max-SAT. Les travaux et résultats présentés dans ce chapitre ont été précédemment publiés [AH14d, AH15b].

Nous discutons dans la première partie de ce chapitre des avantages et inconvénients du schéma MPS et décrivons son implémentation dans notre solveur AHMAXSAT. Nous montrons dans la deuxième partie comment les informations disponibles dans ce schéma peuvent être utilisées pour influer sur la structure des sous-ensembles inconsistants détectés lors du calcul de la borne inférieure. Nous proposons en particulier une heuristique visant à réduire leur taille. La troisième partie est consacrée à la présentation des résultats de l'étude expérimentale que nous avons conduite, dans laquelle nous évaluons l'apport du schéma MPS et comparons différentes heuristiques de construction des sous-ensembles inconsistants.

3.1 Schéma basé sur les sources de propagation multiples

Comme nous l'avons vu dans la première partie de cette thèse, les solveurs récents de type séparation et évaluation suppriment fréquemment des clauses de la formule, ce qui provoque la désaffectation des variables propagées par ces clauses. Dans le schéma FPS, si la première source de propagation d'une variable x est supprimée alors x est désaffectée. Toutes les propagations faites après celle de x doivent également être défaites pour garantir que les sources de propagation de x préalablement ignorées soient maintenant considérées. Parmi les propagations défaites, certaines peuvent toujours avoir des sources de propagation valides, même après avoir désaffecté x. Ces variables seront donc immédiatement réaffectées. Le schéma FPS peut donc conduire à désaffecter et réaffecter des variables inutilement. L'exemple suivant illustre cette situation.

Exemple 11. Considérons la formule non-valuée $\Phi_1 = \{c_1, \ldots, c_{10}\}$ avec $c_1 = \{x_1\}$, $c_2 = \{\overline{x}_1, x_2\}$, $c_3 = \{\overline{x}_1, \overline{x}_2, x_3\}$, $c_4 = \{\overline{x}_2, x_4\}$, $c_5 = \{x_5\}$, $c_6 = \{\overline{x}_5, x_2\}$, $c_7 = \{x_6\}$, $c_8 = \{\overline{x}_6, x_7\}$, $c_9 = \{\overline{x}_6, x_3\}$ et $c_{10} = \{\overline{x}_6, \overline{x}_7, \overline{x}_3\}$. L'application de SUP* sur la formule Φ_1 conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5, x_6@c_7, x_7@c_8\rangle$ (x_1 est propagé par la clause c_1, puis x_2 par c_2, etc.). La clause c_{10} est devenue vide. La fi-

gure 3.1 montre le graphe d'implications correspondant à cette séquence de propagations. On peut remarquer que les clauses c_6 et c_9, qui sont des sources de propagation (mais pas les premières) de respectivement x_2 et x_3 ne sont pas représentées dans le graphe d'implications. L'ensemble des clauses $\psi_1 = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ qui ont menées par propagation unitaire au conflit (i.e. à rendre la clauses c_{10} vide) est un sous-ensemble inconsistant de Φ_1. Si on supprime les clauses de ψ_1 de la formule, alors toutes les propagations causées par ces clauses doivent être défaites. La plus ancienne est $x_1@c_1$ et comme les propagations sont défaites dans l'ordre chronologique inverse dans le schéma FPS, toutes les propagations sont défaites. On obtient la formule $\Phi'_1 = \{c_4, c_5, c_6, c_9\}$ et l'application de SUP* sur Φ'_1 conduit à la séquence de propagations $\langle x_5@c_5, x_2@c_6, x_4@c_4\rangle$. On peut noter que ces trois variables ont été désaffectées puis réaffectées consécutivement.

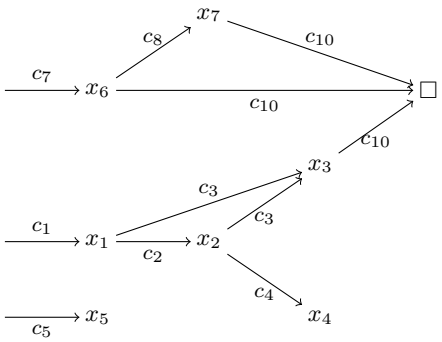


FIGURE 3.1 – Graphe d'implications de la formule Φ_1 de l'exemple 11. On peut noter que les clauses c_6 et c_9 qui ne sont pas les premières sources de propagation de respectivement x_2 et x_3 ne sont pas représentées.

Pour palier cette situation, nous proposons un nouveau schéma de propagation considérant toutes les sources de propagation des variables (*multiple propagation sources scheme*, MPS). Un tel schéma change profondément la manière dont la propagation unitaire est appliquée. Toutes les sources de propagation de chaque variable sont mémorisées et les variables propagées ne sont désaffectées que lorsqu'elles n'ont plus aucune source de propagation. Ainsi, les propagations peuvent être défaites de manière non-chronologique et cela évite les opérations consécutives de désaffectation et réaffectation. Les conflits sont également détectés plus tôt que dans le schéma FPS. Lorsqu'une variable à des sources de propagation pour ses deux polarités, alors l'une d'entre elles sera nécessairement falsifiée lorsqu'on affectera la variable.

Dans le reste de cette section, nous définissons les bases du nouveau schéma de propagation avant de discuter de ses avantages, inconvénients, et de son implémentation dans notre solveur AHMAXSAT.

3.1.1 Graphe d'implications plein

Dans le schéma de propagation FPS, les séquences de propagations sont généralement modélisées sous la forme d'un graphe d'implications (cf. définition 19) : chaque variable propagée est représentée par un nœud et les arcs relient les raisons des propagations

(les littéraux falsifiés des clauses sources) à leurs conséquences (les littéraux propagés). Une telle représentation n'est pas adaptée au schéma de propagation MPS car elle ne permet pas de faire la distinction entre les littéraux falsifiés de chacune des clauses unitaires provoquant la propagation d'une variable. Nous définissons donc une nouvelle modélisation pour représenter les étapes de propagations réalisées dans le schéma MPS.

Définition 34 (Graphe d'implications plein)**.** Soit $\Phi = \{c_1, \ldots, c_m\}$ une formule définie sur un ensemble de variables propositionnelles $X = \{x_1, \ldots, x_n\}$ et I une interprétation (partielle ou complète) des variables de X. Nous supposons qu'il ne peut y avoir qu'une seule clause falsifiée (i.e. UP est arrêtée dès qu'une clause vide est générée). Un graphe d'implications plein est un graphe ET/OU orienté et sans circuit $G = (V_{ou}, V_{et}, A)$ avec V_{ou} l'ensemble des nœuds OU qui représentent les variables affectées et V_{et} l'ensemble des nœuds ET qui représentent les sources de propagation (les clauses unitaires) et A est l'ensemble des arcs qui lient les clauses unitaires aux variables propagées et les variables affectées aux clauses qu'elles réduisent. Formellement :

$$\begin{aligned} V_{ou} &= \{l \in I\} \\ V_{et} &= \{c_k \in \Phi \text{ t.q. } \exists l \in I,\ c_k \in src(l)\} \\ A &= \{(l, c_p) \text{ t.q. } c_p \in V_{et} \text{ est réduit par } l \in I\} \\ &\quad\ \{(c_q, l) \text{ t.q. } c_q \in V_{et} \text{ et } c_q \in src(l)\} \end{aligned}$$

L'exemple qui suit illustre le fonctionnement du schéma MPS.

Exemple 12. Considérons à nouveau la formule Φ_1 définie dans l'exemple 11. L'application de SUP* dans le schéma MPS produit les six mêmes étapes de propagation que dans l'exemple 11 : $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5, x_6@c_7\rangle$. La variable x_7 a deux sources de propagation de polarités opposées, c_8 et c_{10}, qui ne peuvent être toutes les deux satisfaites. La figure 3.2 montre le graphe d'implications plein modélisant cette situation. On peut noter que les secondes sources de propagation de x_2 et x_3 (respectivement c_6 et c_9) sont représentées dans le graphe d'implications plein.

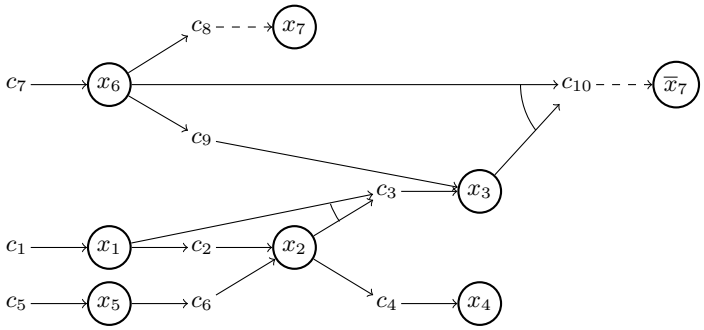


FIGURE 3.2 – Graphe d'implications plein de la formule Φ_1 de l'exemple 12. Les nœuds cerclés sont les variables propagées tandis que les nœuds non cerclés sont les sources de propagation (les clauses unitaires). On peut remarquer que les sources de propagation c_6 de x_2 et c_9 de x_3 sont représentées.

Comme dans l'exemple précédent, on peut construire le sous-ensemble inconsistant $\psi_1 = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$ en prenant la première source de propagation de chacune

des variables affectées ayant mené au conflit. Si les clauses de ψ_1 sont supprimées de la formule, alors les variables x_1, x_3, x_6 et x_7 n'ont plus de source de propagation et doivent donc être désaffectées. On peut noter que les variables x_5, x_2 et x_4 restent propagées car elles ont toujours au moins une source de propagation.

3.1.2 Problème des circuits

Le schéma de propagation basé sur les sources de propagation multiples a un inconvénient : des circuits peuvent apparaître dans le graphe d'implications plein. Un circuit apparaît lorsqu'un littéral propagé l_i conduit, par une ou plusieurs étapes de propagation unitaire, à rendre unitaire une clause qui est une source de propagation de l_i. Dans cette situation, quand toutes les autres sources de propagation de l_i sont supprimées, l_i reste propagé. Cependant, falsifier l_i ne falsifie aucune clause. Cela conduirait à défaire toutes les propagations dépendantes de l_i et donc la dernière source de propagation de l_i ne serait plus unitaire. Donc, la prise en compte de ces circuits mènerait à la détection de faux conflits.

Exemple 13. Considérons une formule non valuée $\Phi_2 = \{c_1, c_2, \ldots, c_6\}$ avec $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x}_1, x_3\}$, $c_4 = \{\overline{x}_3, x_1\}$, $c_5 = \{\overline{x}_3, x_4\}$ et $c_6 = \{\overline{x}_2, \overline{x}_4\}$. Après application de la propagation unitaire simulée sur Φ_2, la variable x_4 a des sources de propagation des deux polarités. Φ_2 est donc inconsistante. Le graphe d'implications plein décrivant les étapes de propagation unitaire réalisées est présenté dans la figure 3.3.

Si on supprime la clause c_1 on obtient la formule $\Phi'_2 = \Phi_2 \setminus \{c_1\}$. La variable x_1 est toujours propagée puisqu'elle a toujours une source de propagation valide c_4 et le conflit est toujours présent dans le graphe d'implications plein (figure 3.4). Cependant, Φ'_2 n'est pas inconsistante et l'interprétation $I = \{\overline{x_1}, x_2, \overline{x_3}, \overline{x_4}\}$ satisfait toutes ses clauses.

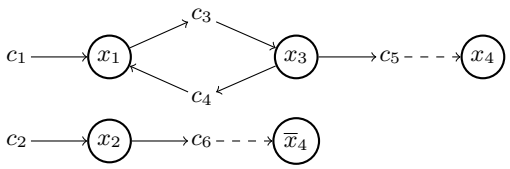


FIGURE 3.3 – Graphe d'implications plein de la formule Φ_2 de l'exemple 13. Un circuit est présent entre les variables propagées x_1 et x_3, mais la propagation de x_1 est valide grâce à son autre source de propagation c_1.

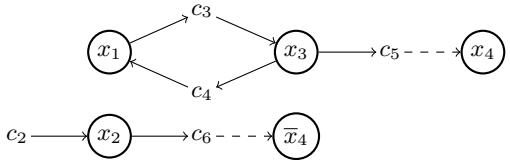


FIGURE 3.4 – Graphe d'implications plein de la formule Φ'_2 de l'exemple 13. Un circuit est toujours présent entre les variables propagées x_1 et x_3, mais cette fois la propagation de x_1 n'est pas valide : elle dépend uniquement du circuit.

Une solution pour éviter la détection de faux conflits consisterait à ignorer les sources de propagations pouvant créer des circuits. Une manière simple de le faire est de considérer le niveau (c'est-à-dire la distance aux sources) des variables et des clauses dans le graphe d'implications plein et d'ignorer les arcs revenant en arrière.

Définition 35 (Niveau dans un graphe d'implications plein)**.** Le niveau d'un littéral l_i dans un graphe d'implications plein peut être définie comme suit :

$$niveau(l_i) \ = \begin{cases} 0, \text{ si } l_i \text{ est une décision} \\ \max(\{niveau(c) : c \in sources(l_i)\}), \text{ si } l_i \text{ est une propagation} \\ +\infty, \text{ sinon} \end{cases}$$

On peut définir le niveau d'une clause c_j comme :

$$niveau(c_j) \ = \ \max(\{niveau(l) : \bar{l} \in c_j \text{ et } l \text{ est satisfait } \}) + 1$$

Proposition 1. Une source de propagation c_j d'une variable x_i ne peut créer de circuit si $niveau(x_i) \geq niveau(c_j)$.

Démonstration. Triviale par la définition des niveaux. □

Quand le niveau d'une nouvelle source de propagation est plus élevé que celui de la variable qu'elle propage, alors elle est simplement ignorée. De cette manière, et pour un coût réduit en termes de temps d'exécution, les solveurs peuvent utiliser le schéma MPS sans l'inconvénient des circuits et des détections de faux conflits qu'ils entraînent. Ce filtrage basé sur le niveau conduit cependant à ignorer des sources de propagation valides. Nous avons testé d'autres alternatives, comme la détection des circuits par exploration du graphe d'implications plein lorsqu'une nouvelle source de propagation de niveau supérieur est ajoutée ou lorsqu'une variable propagée n'a plus de sources de propagation de niveau inférieur (pour éviter de défaire la propagation quand ce n'est pas nécessaire). Dans les deux cas, le temps de calcul passé à analyser le graphe d'implications était supérieur aux gains apportés.

3.1.3 Implémentation et complexité

Nous avons implémenté le schéma de propagation MPS comme un composant central de notre solveur AHMAXSAT. Chaque variable a deux listes allouées statiquement pour stocker ses sources de propagation positives et négatives. Quand une clause c_j devient unitaire, AHMAXSAT calcule son niveau et identifie le littéral l_i qu'elle propage (le seul littéral non falsifié de c_j). Si le niveau de c_j est inférieur à celui de l_i, c_j est ajoutée à la liste des sources de propagation de $var(l_i)$. Lorsqu'une variable est affectée, son niveau est calculé en examinant ses sources de propagation. Quand une variable a des sources de propagation des deux polarités, AHMAXSAT construit un sous-ensemble inconsistant en analysant le graphe d'implications plein et applique un traitement sur ses clauses comme le font les autres solveurs de type séparation et évaluation (cf. section 2.3.4). Les variables propagées sont défaites lorsqu'elles n'ont plus aucune source de propagation.

Le schéma de propagation basé sur les sources multiples de propagation ne change pas la complexité de notre algorithme. Puisqu'il ne peut y avoir plus de sources de propagation que de clauses dans la formule, la complexité dans le pire des cas du mécanisme

de détection et de traitement des sous-ensembles inconsistants (propagation unitaire simulée, littéraux contradictoire et transformations des sous-ensembles inconsistants) reste inchangée. En pratique cependant, plus de clauses seront examinées durant l'application de la propagation unitaire et lors de la construction des sous-ensembles inconsistants. De plus, les structures de données utilisées par le solveur doivent être adaptées pour supporter les propagations non-chronologiques.

3.2 Réduction de la taille des sous-ensembles inconsistants

Nous montrons dans cette section comment les informations disponibles dans le schéma MPS peuvent être utilisées pour influer sur les caractéristiques des sous-ensembles inconsistants construits lors du calcul de la borne inférieure. La structure et les propriétés (nombre de clauses, tailles des clauses, etc.) des sous-ensembles inconsistants générés lors de l'analyse des conflits ont un impact important sur la capacité des solveurs à détecter d'autres sous-ensembles inconsistants dans la formule. En particulier, réduire la taille des sous-ensembles inconsistants laisse plus de clauses disponibles dans la suite de la recherche, clauses qui peuvent être utilisées pour appliquer la propagation unitaire et donc pour détecter d'autres sous-ensembles inconsistants. Cela peut permettre d'améliorer la qualité de l'estimation de la borne inférieure et donc réduire le nombre de nœuds de l'arbre de recherche qui seront explorés.

Quand un solveur utilisant le schéma FPS détecte un conflit (une clause vide) lors du calcul de la borne inférieure (algorithme 2.1, ligne 7), il construit un sous-ensemble inconsistant en analysant la séquence de propagations qui a mené au conflit (algorithme 2.2, ligne 6). L'algorithme 3.1 décrit cette procédure. Une liste Q est utilisée pour stocker les littéraux à analyser. Initialement, Q contient les littéraux réduisant la clause falsifiée c (ligne 3). Puis l'algorithme traite les littéraux de Q dans l'ordre inverse des propagations (ligne 5). Pour chaque littéral l, il sélectionne la première source de propagation c de l (ligne 6). Puis, il ajoute c au sous-ensemble inconsistant ψ et les littéraux propagés réduisant c à Q (lignes 7-8). Ce processus est répété jusqu'à ce que tous les littéraux de Q aient été traités. Comme seule la première source de propagation de chaque littéral est connue, un seul sous-ensemble inconsistant peut être construit.

Algorithme 3.1 : Fonction `construction_sous_ensemble_inconsistant` dans un schéma FPS.

Data : Une formule Φ, une interprétation I et une clause falsifiée c.
Result : ψ un sous-ensemble inconsistant de Φ.

```
1 begin
2     ψ ← {c};
3     Q ← {littéraux propagés réduisant c};
4     while Q ≠ ∅ do
5         l ←littéral propagé le plus récemment de Q;
6         c ← src(l);
7         ψ ← ψ ∪ {c};
8         Q ← (Q \ {l}) ∪ {littéraux propagés réduisant c};
9     return ψ;
```

Propriété 3. Pour une formule Φ en entrée définie sur un ensemble de variables propositionnelles X et une interprétation I, la complexité temporelle dans le pire des cas de l'algorithme 3.1 est en $\mathcal{O}(v * k)$ avec $v = |X| - |I|$ le nombre de variables non affectées de la formule Φ et k le nombre maximal de littéraux par clause.

Démonstration. Dans le pire des cas, toutes les variables non affectées par des décisions ou des propagations unitaires réelles ont été affectées lors de l'application de la propagation unitaire simulée. On a donc au plus $v = |X| - |I|$ variables propagées. Lors de l'analyse de chacune des sources de propagation des ces v variables, le nombre de littéraux examinés ne peut dépasser le nombre maximal de littéraux par clause k. La complexité temporelle dans le pire des cas de l'algorithme est donc bien en $\mathcal{O}(v * k)$. □

A l'inverse, un solveur utilisant le schéma MPS peut choisir, pour chaque propagation ayant mené au conflit, la source de propagation qu'il ajoute au sous-ensemble inconsistant. Donc un tel solveur peut influer sur la structure et les propriétés des sous-ensembles inconsistants qu'il produit. Nous proposons une heuristique simple visant à réduire la taille des sous-ensembles inconsistants. Cette heuristique est présentée dans l'algorithme 3.2. Pour chaque littéral l apparaissant dans la séquence de propagations, l'heuristique ajoute à l'ensemble inconsistant ψ la clause source de propagation de l qui ajoute le moins de nouveaux littéraux à l'ensemble Q, c'est-à-dire la clause c qui minimise $|c \setminus (\{l\} \cup (c \cap Q))|$. La fonction suivante est utilisée dans l'algorithme : pour deux ensembles d'ensemble de littéraux A et B, `plus_petit_couple(A,B)` renvoie un couple (a, b) tel que $a \in A$, $b \in B$ et $\forall (a', b') \in A \times B$, $|a \cup b| \leq |a' \cup b'|$. Nous appellerons cette heuristique SIS (*smaller inconsistent subset*). On peut noter que si l'heuristique SIS peut permettre de réduire la taille des sous-ensembles inconsistants qu'elle construit, elle ne garantie pas que cette taille soit minimale.

Algorithme 3.2 : Fonction `construction_sous_ensemble_inconsistant` dans un schéma MPS avec l'heuristique SIS.

Data : Une formule Φ, une interprétation I et une variable x ayant des sources de propagation des deux polarités.
Result : ψ un sous-ensemble inconsistant de Φ.

```
1  begin
2      (c1, c2) ← plus_petit_couple(src(x),src(x̄));
3      ψ ← {c1, c2};
4      Q ← {littéraux propagés réduisant c1} ∪ {littéraux propagés réduisant c2};
5      while Q ≠ ∅ do
6          l ←littéral propagé le plus récemment de Q;
7          (Q, c) ← plus_petit_couple({Q}, src(l));
8          ψ ← ψ ∪ {c};
9          Q ← Q ∪ {littéraux propagés réduisant c/{x}};
10     return ψ;
```

Propriété 4. Pour une formule Φ en entrée définie sur un ensemble de variables propositionnelles X et une interprétation I, la complexité temporelle dans le pire des cas de l'algorithme 3.2 est en $\mathcal{O}(m * k)$ avec m le nombre de clauses de Φ qui ne sont pas vide sous l'interprétation I et k le nombre maximal de littéraux par clause.

Démonstration. Pour une variable propagée donnée, l'examen de ses sources de propagation peut être réalisé en temps linéaire par rapport à la somme des littéraux qu'elles contiennent. Pour chaque source de propagation, au plus k littéraux seront donc examinés (avec k la taille maximale des clauses de la formule). De la même manière, la complexité temporelle dans le pire des cas de la fonction `plus_petit_couple` est linéaire par rapport à la somme des littéraux des clauses des deux ensembles passés en argument. Comme le nombre total de sources de propagation ne peut excéder le nombre de clauses de la formule m (une clause ne pouvant propager qu'une seule variable), la complexité temporelle de l'algorithme est bien en $\mathcal{O}(m * k)$. □

Exemple 14. Considérons à nouveau la formule Φ_1 de l'exemple 11. Nous avons vu que dans un schéma de propagation de type FPS (cf. figure 3.1), le sous-ensemble inconsistant de Φ_1 qui peut être construit en ne considérant que les premières sources de propagation des variables est $\psi_1 = \{c_1, c_2, c_3, c_7, c_8, c_{10}\}$.

Dans un schéma de propagation de type MPS (cf. figure 3.2), l'heuristique SIS fonctionne comme suit. Elle commence par ajouter à l'ensemble Q les littéraux falsifiés (x_3 et x_7) des sources de propagations de chaque polarité (c_8 et c_{10}) de la variable conflictuelle x_7. Puis il choisit le littéral de $Q = \{x_3, x_6\}$ propagé en dernier, x_3, qui a deux sources de propagation c_3 et c_9. L'heuristique choisit d'ajouter au sous-ensemble inconsistant la clause c_9 qui contient le moins de variables n'apparaissant pas déjà dans Q (0 variables) que c_3 (2 variables, x_1 et x_2). Il reste une seule variable dans Q, x_6, qui a une seule source de propagation c_7. Cette dernière n'a aucun littéral falsifié. L'ensemble inconsistant construit par l'heuristique est donc $\psi_2 = \{c_7, c_8, c_9, c_{10}\}$. On peut observer que ψ_2 contient deux clauses de moins que ψ_1. Ces clauses peuvent être utilisées pour appliquer SUP et potentiellement permettre de détecter d'autres sous-ensembles inconsistants.

3.3 Étude expérimentale

Le schéma de propagation est un composant important des solveurs de type séparation et évaluation. Il conditionne la manière dont les propagations sont faites et défaites et la manière dont sont analysés les conflits. Les structures de données doivent être adaptées en conséquence. Dans notre solveur AHMAXSAT, environ 40 à 50% du code source est affecté directement ou indirectement par ce schéma et le temps passé à appliquer la propagation unitaire représente une part importante du temps d'exécution total du solveur.

Comme dit précédemment, le schéma MPS est un des composants centraux de notre solveur AHMAXSAT, qui a été construit autour de ce schéma. Implémenter une variante d'AHMAXSAT utilisant le schéma FPS demanderait donc une réécriture importante du code de notre solveur. Les performances d'une telle variante dépendraient fortement de la qualité de cette réécriture et du temps passé à l'optimiser. Nous avons donc choisi d'évaluer le potentiel et l'impact du schéma MPS sur les performances du solveur par deux séries d'expérimentations. Dans la première, nous avons cherché à évaluer le pourcentage d'étapes de propagation unitaire que le schéma MPS permet d'éviter. Dans la seconde série de tests, nous avons comparé l'heuristique SIS avec deux heuristiques de construction des sous-ensembles inconsistants simulant le comportement du schéma FPS. Tous les résultats présentés dans cette section ont été obtenus en suivant le protocole expérimental décrit dans la section 2.6.

3.3.1 Mesure des étapes de propagation évitées

Nous avons tout d'abord cherché à estimer le pourcentage d'étapes de propagation inutiles évitées grâce au schéma MPS. Dans une variante dédiée de notre solveur, nous avons maintenu parallèlement aux structures de données du schéma MPS une liste chronologique des propagations effectuées par AHMAXSAT, comme le ferait un solveur utilisant le schéma FPS. Nous avons utilisé cette variante pour estimer les étapes de propagations évitées en distinguant deux situations.

Quand la première source de propagation d'une variable est supprimée, la variable est désaffectée dans le schéma FPS. Dans notre solveur, si la variable a d'autres sources de propagation, alors elle n'est pas désaffectée. C'est ce que nous appellerons les propagations évitées "directes".

Nous avons également estimé le nombre d'étapes de propagation inutiles qui auraient été causées par la désaffectation par ordre chronologique inverse. Lorsque des variables propagées sont désaffectées, nous avons mesuré le nombre de variables restant propagées et situées après la première des variables défaites dans l'ordre chronologiques des propagations. Nous les appellerons propagations évitées "indirectes".

Pour comprendre l'impact des résultats, il est important de rappeler que la propagation unitaire est utilisée très fréquemment dans les solveurs de type séparation et évaluation. Par exemple, AHMAXSAT fait en moyenne 1500 étapes de propagations à chaque nœud de l'arbre de recherche et le nombre total moyen de propagations par instance est d'environ 150 millions. Le pourcentage d'étapes de propagation évitées est montré dans la table 3.1 (colonne PE). En moyenne, le schéma MPS permet de réduire de 28% le nombre de propagations réalisées par AHMAXSAT. On peut remarquer que ce pourcentage varie fortement d'un classe d'instances à une autre et peut aller jusqu'à 96% (sur la classe d'instance wpms/crafted/set-covering). Il est également intéressant de noter que les propagations évitées directes sont en moyenne inférieures à 1%, tandis que les indirectes sont supérieures à 27%. Enfin, il convient de rappeler que ces chiffres sont une estimation et le nombre réel de propagations évitées par rapport à un schéma FPS peut varier. De plus, ces résultats ne permettent pas de présumer précisément de l'impact du schéma MPS sur le temps de résolution du solveur. Le gain dû aux étapes de propagation évitées peut être compensé en partie par le temps consacré au maintien des sources multiples de propagation.

3.3.2 Heuristique de construction des sous-ensembles inconsistants

Dans la deuxième partie de cette étude expérimentale, nous avons cherché à évaluer l'impact de l'heuristique de construction des sous-ensembles inconsistants présentée en section 3.2 sur les performances d'AHMAXSAT. Pour ce faire, nous la comparons à deux heuristiques simples qui simulent les sous-ensembles inconsistants pouvant être obtenus dans un schéma de type FPS. Nous considérons les trois variantes suivantes, qui se distinguent par la manière dont elles choisissent les clauses à ajouter aux sous-ensembles inconsistants :

- AHMAXSAT^{SIS} choisit les sources de propagation à ajouter aux sous-ensembles inconsistants selon l'heuristique SIS,
- AHMAXSAT^{F} choisit la première source de propagation,
- AHMAXSAT^{R} choisit aléatoirement parmi les sources de propagation.

TABLE 3.1 – Comparaison des heuristiques de construction des sous-ensembles inconsistants dans AHMAXSAT. Les deux premières colonnes montrent les classes d'instances et le nombre d'instances par classe. La colonne PE donne l'estimation du pourcentage d'étapes de propagation inutiles évitées grâce au schéma MPS et la colonne SP donne le nombre moyen de sources de propagation des variables lors de l'analyse des graphes d'implications. Pour chaque variante d'AHMAXSAT, les colonnes S, D et T donnent respectivement le nombre d'instances résolues, le nombre moyens de nœuds explorés et le temps moyen d'exécution. Les colonnes marquées avec une étoile ne prennent en compte que les instances résolues par toutes les variantes.

instances classes			#	PE	SP	AHMAXSAT^{SIS}			AHMAXSAT^{R}			AHMAXSAT^{F}		
						S	D*	T*	S	D*	T*	S	D*	T*
ms	crafted	bipartite	100	18%	1.07	100	34915	**90,7**	100	37118	98	100	37099	96,6
		maxcut	67	9%	1.14	56	171380	**44,7**	56	172639	48,9	56	181242	47,8
		set-covering	10	-	-	0	-	-	0	-	-	0	-	-
	random	highgirth	82	14%	1.06	7	5016310	**1166**	7	5037575	1174,3	7	5075143	1171,5
		max2sat	100	12%	1.05	100	45444	**89,5**	100	47143	93,1	100	47318	93,7
		max3sat	100	7%	1.06	100	340968	**231,2**	100	351372	240,2	100	351199	237
		min2sat	96	10%	1.04	96	1046	**2,6**	96	1147	2,9	96	1077	2,7
pms	crafted	frb	25	3%	1.03	5	585949	**154,2**	5	591410	156,7	5	601020	158,3
		job-shop	3	-	-	0	-	-	0	-	-	0	-	-
		maxclique	158	2%	1.03	133	23107	30,2	133	21831	**29,8**	133	23136	31
		maxone	140	61%	1.15	109	133245	41,6	**110**	68842	**25,7**	109	152188	46,2
		min-enc/kbtree	42	39%	1.14	**34**	166705	**396,6**	31	203516	523,1	**34**	193609	458,1
		pseudo/miplib	4	5%	1.11	2	288	$< 0,1$	2	272	$< 0,1$	2	281	$< 0,1$
		reversi	44	43%	1.30	8	4555	**129,2**	8	5697	163,5	8	5193	150,3
		scheduling	5	-	-	0	-	-	0	-	-	0	-	-
	random	min2sat	60	21%	1.05	58	12100	**186,8**	58	12607	192,1	58	12468	190,9
		min3sat	60	4%	1.04	58	91569	**249,9**	58	99796	270,1	58	100013	272,9
		pmax2sat	60	28%	1.14	60	1021	**3,2**	60	1128	3,7	60	1200	3,8
		pmax3sat/hi	30	9%	1.06	30	84944	**60,1**	30	87072	62,4	30	86944	61,5
wpms	crafted	auction	40	53%	1.57	40	310021	**115,5**	40	309489	136,1	40	289406	120,3
		CSG	10	32%	2.13	4	54813	452,3	4	50642	**421,3**	4	54330	432,3
		frb	34	3%	1.06	14	212369	**58,7**	14	214290	58,9	14	217815	64,3
		min-enc	74	76%	1.74	**64**	17754	**61,9**	55	11052	66,7	**64**	9547	65
		pseudo/miplib	12	63%	2.31	4	1505	**158,8**	4	1730	194,5	4	1570	172,4
		ramsey	15	30%	1.45	4	157559	**44,8**	4	157725	62,7	4	158851	50,5
		random-net	32	80%	1.70	**3**	335161	**237,5**	2	702456	580,1	1	2791162	1325,1
		set-covering	45	96%	1.07	**25**	3771	46,5	24	3837	45,3	**25**	3856	**45**
		wmaxcut	48	33%	1.44	46	26985	**49,4**	46	27192	63,9	46	28224	58,1
	random	wmax2sat	120	44%	1.20	120	3993	**45,4**	120	4875	59,6	120	4364	51,4
		wmax3sat	40	37%	1.15	40	36836	**99,5**	40	37820	106,8	40	37768	104,3
		wpmax2sat	90	55%	1.49	90	575	**5,5**	90	615	7,2	90	601	6,5
		wpmax3sat/hi	30	41%	1.15	30	30451	**84,1**	30	31230	90,5	30	31219	88,5
Ensemble			1776	28%	1.17	**1440**	100391	**91**	1427	97905	98,6	1438	105565	97,4

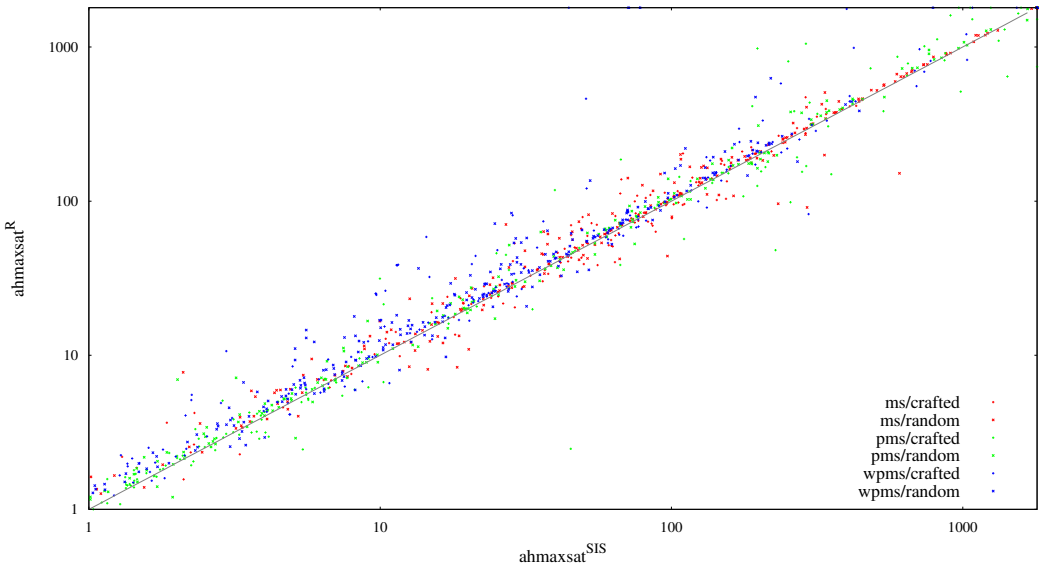
Les résultats globaux sont présentés dans la table 3.1. On peut voir que la variante utilisant l'heuristique SIS résout plus d'instances (respectivement +13 et +2 par rapport à AHMAXSAT^R et AHMAXSAT^F, colonnes S) et son temps moyen de résolution est plus faible (respectivement -7,8% et -6,6%, colonnes T). Ces résultats varient cependant fortement d'une classe d'instances à une autre. Par exemple, sur les instances wpms/crafted/random-net, le temps moyen de résolution est divisé par deux par rapport à celui d'AHMAXSAT^R et par 5 par rapport à AHMAXSAT^F. A l'inverse, sur les instances de la catégorie pms/crafted/maxone, le temps moyen de résolution d'AHMAXSAT^{SIS} est sensiblement plus élevé que celui d'AHMAXSAT^R.

La figure 3.5 compare instance par instance le temps de résolution d'AHMAXSAT^{SIS} et celui des autres variantes. Cela vient confirmer les résultats précédemment observés : le temps de résolution d'AHMAXSAT^{SIS} est meilleur que celui des autres variantes sur une majorité d'instances, mais de fortes disparités existent entre les catégories d'instances et au sein même de ces catégories.

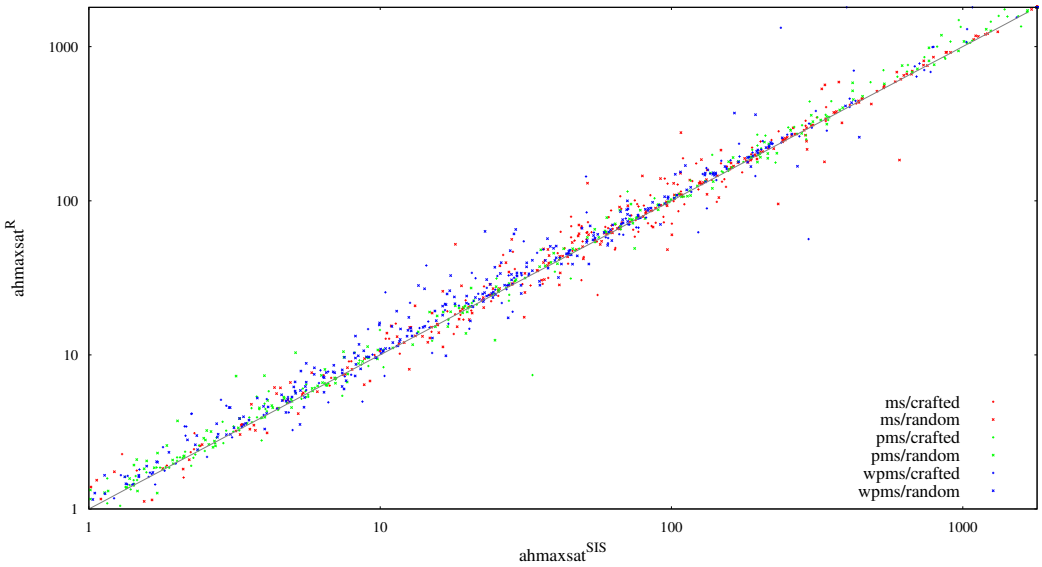
De manière surprenante, le nombre moyen de nœuds explorés par AHMAXSAT^{SIS} est légèrement plus élevé que celui d'AHMAXSAT^R (table 3.1, colonnes D). Ceci s'explique en partie par les importantes différences observables sur quelques classes d'instances (e.g. pms/crafted/maxone) qui influe fortement sur la moyenne. La comparaison instance par instance du nombre de nœuds explorés par AHMAXSAT^{SIS} et celui des autres variantes (figure 3.6) montre que le nombre de nœuds explorés est globalement similaire pour toutes les variantes. On peut en déduire que le gain en temps de résolution apporté par l'heuristique SIS provient davantage d'une amélioration de la vitesse de parcours de l'arbre de recherche (induite par la réduction de la taille des sous-ensembles inconsistants) que de la réduction du nombre de nœuds explorés.

Le gain en terme de temps moyen de résolution apporté par l'heuristique SIS, bien que significatif, est limité. Plusieurs facteurs peuvent expliquer cela. Tout d'abord, comme nous l'avons dit précédemment, l'heuristique SIS fonctionne de manière naïve : le choix des sources de propagation à ajouter aux sous-ensembles inconsistants se base uniquement sur le nombre de nouvelles variables ajoutées au sous-ensemble inconsistant par rapport à celles déjà rencontrées. Elle ne considère pas le nombre de clauses ayant mené à la propagation de ces variables, qui seront également ajoutées au sous-ensemble inconsistant. Par conséquent, rien ne garantie la minimalité des sous-ensembles inconsistants générés. Une heuristique plus complexe pourrait permette de réduire plus efficacement la taille des sous-ensembles inconsistants générés.

Ensuite, l'impact des propriétés des sous-ensembles inconsistants sur le comportement des solveurs de type séparation et évaluation est encore largement méconnu. S'il est généralement admis que des sous-ensembles inconsistants de petite taille permettent de laisser plus de clauses disponibles pour appliquer la propagation unitaire, d'autres points n'ont à notre connaissance jamais été étudiés. On peut par exemple citer le rôle que peuvent avoir les clauses dures dans les sous-ensembles inconsistants, ou l'impact des transformations par max-résolution sur le reste de la recherche et en particulier sur la propagation unitaire. Une étude approfondie de ces éléments pourrait permettre de développer des heuristiques de construction des sous-ensembles inconsistants plus performantes.

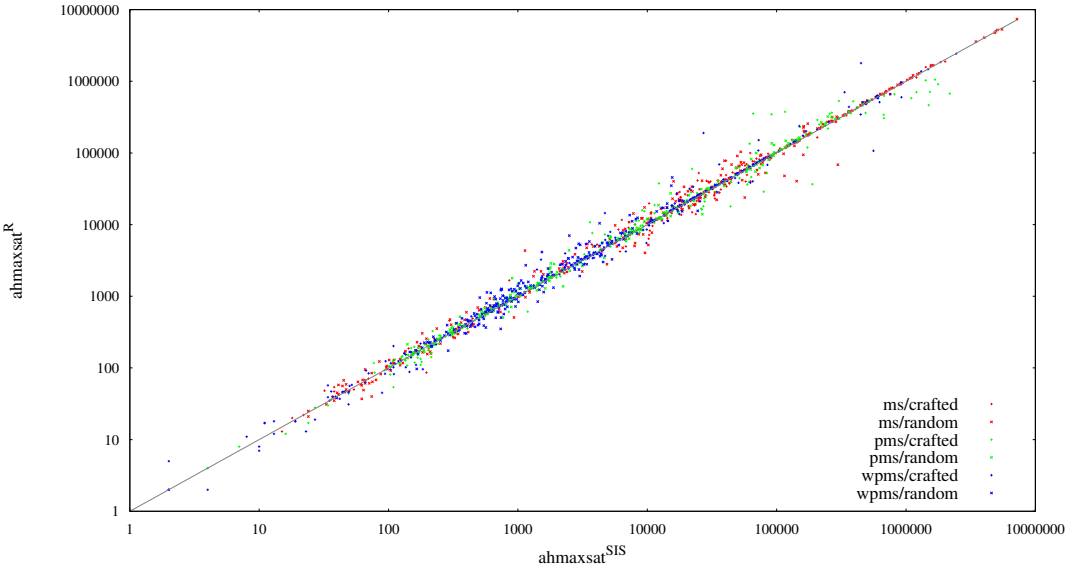


(a) AHMAXSAT^{SIS} vs. AHMAXSAT^{R}

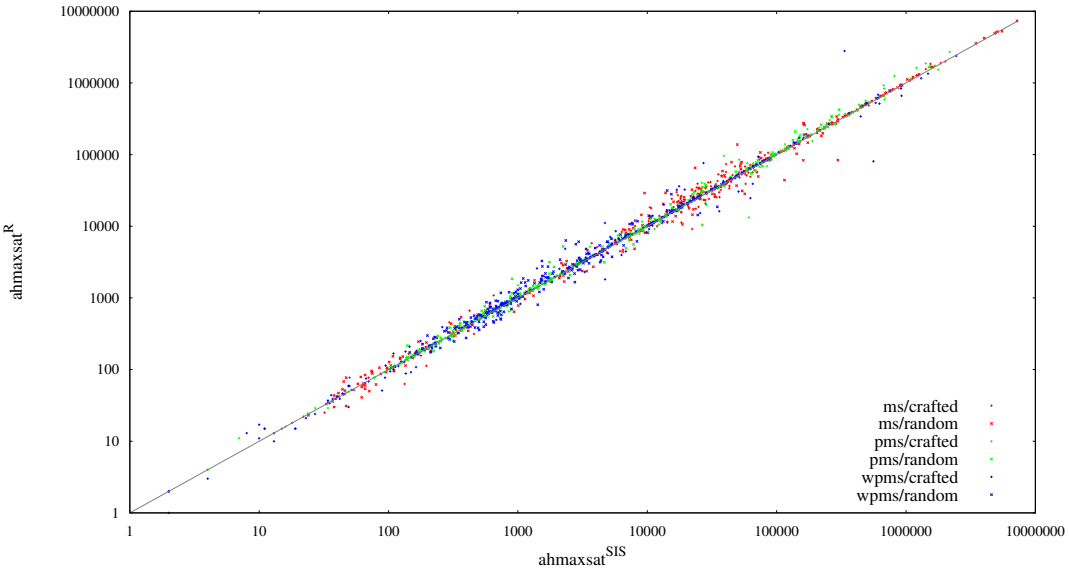


(b) AHMAXSAT^{SIS} vs. AHMAXSAT^{F}

FIGURE 3.5 – Comparaison détaillée des temps de résolution des variantes d'AHMAXSAT. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.



(a) AHMAXSAT^{SIS} vs. AHMAXSAT^{R}



(b) AHMAXSAT^{SIS} vs. AHMAXSAT^{F}

FIGURE 3.6 – Comparaison détaillée du nombre de nœuds explorés par les variantes d'AHMAXSAT. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

3.4 Conclusion

Nous avons présenté dans ce chapitre un nouveau schéma de propagation qui prend en compte toutes les sources de propagation des variables. Nous avons montré comment ce schéma pouvait être utilisé dans les solveurs de type séparation et évaluation pour défaire les propagations de manière non-chronologique, ce qui permet de réduire le nombre d'étapes de propagation réalisées par les solveurs. Les résultats de l'étude expérimentale que nous avons conduite confirme l'utilité de ce nouveau schéma dans notre solveur AHMAXSAT.

Nous pensons que l'intérêt de ce nouveau schéma n'est pas limité aux solveurs de type séparation et évaluation pour Max-SAT. Toute application de la propagation unitaire de manière non-chronologique (i.e. telle que les propagations ne sont pas défaites dans l'ordre inverse des propagations) entraînera dans le schéma classique de nombreuse opérations de propagation inutiles. Notre schéma pourrait donc permettre d'étendre le champs d'application de la propagation unitaire à d'autres types de solveurs explorant l'espace des solutions sans construire d'arbre de recherche.

Nous avons également montré que les informations disponibles dans ce nouveau schéma de propagation pouvaient être utilisées pour influer sur la structure des sous-ensembles inconsistants générés par SUP et FL. Nous avons proposé une heuristique simple exploitant ces informations qui vise à réduire la taille des sous-ensembles inconsistants. Les résultats expérimentaux montrent qu'elle permet sur la plupart des classes d'instances considérées de réduire le nombre de nœuds explorés par le solveur et donc le temps de résolution.

Cela conduit à s'interroger sur ce que sont les "bonnes" propriétés des sous-ensembles inconsistants et donc plus généralement sur l'impact de leurs transformations sur le comportement et l'efficacité des solveurs de type séparation et évaluation. Nous poursuivrons dans cette voie dans le prochain chapitre, où nous étudierons les effets de l'ordre d'application des étapes de max-résolution sur le nombre et la taille des clauses de compensation ajoutées lors de la transformation des sous-ensembles inconsistants.

Chapitre 4

Ordre d'application des étapes de max-résolution

À chaque nœud de l'arbre de recherche, les solveurs de type séparation et évaluation pour Max-SAT calculent la borne inférieure en estimant le nombre de sous-ensembles inconsistants disjoints présents dans la formule (cf. algorithmes 2.2) et 2.3). Nous avons vu que les sous-ensembles inconsistants sont détectés par des méthodes basées sur la propagation unitaire (propagation unitaire simulée, SUP, ou littéraux contradictoires, FL) puis traités pour éviter d'être détectés plus d'une fois (cf. section 2.3.4). Deux types de traitements sont appliqués aux sous-ensembles inconsistants par les solveurs récents : la suppression et la transformation par max-résolution.

Nous nous intéressons ici à la transformation par max-résolution qui consiste à appliquer des étapes de max-résolution entre les clauses des sous-ensembles inconsistants prises dans l'ordre inverse des propagations jusqu'à générer une clause résolvante vide. La formule transformée contient, outre la clause résolvante, un ensemble de clauses de compensation ajoutées pour préserver l'équivalence de la formule. Nous avons vu que cette méthode de transformation peut être utilisée comme une forme d'apprentissage pour limiter la redondance dans le calcul de la borne inférieure. Cependant, elle a également des désavantages qui empêchent sa généralisation. Elle n'est par conséquent appliquée que dans certains cas particuliers (cf. section 2.3.4.3).

Nous nous intéressons dans ce chapitre à l'un de ces inconvénients : l'augmentation de la taille de la formule entraînée par l'ajout des clauses de compensation. Nous avons

observé que l'ordre d'application des étapes de max-résolution lors de la transformation des sous-ensembles inconsistants a un impact sur les caractéristiques des clauses de compensation générées. En particulier, pour deux variables voisines dans un graphe d'implication, nous avons montré qu'appliquer la max-résolution d'abord sur celle produisant le plus petit résolvant produit moins de clauses de compensation et qu'elles sont de plus petite taille. Nous avons donc proposé une heuristique de choix de l'ordre d'application de la max-résolution qui privilégie les variables produisant les plus petits résolvantes intermédiaires. L'étude expérimentale menée en intégrant cette heuristique dans AHMAXSAT montre que non seulement le nombre et la taille des clauses de compensation sont effectivement réduits mais également que le comportement de la propagation unitaire sur les formules transformées s'en trouve amélioré.

Nous rappelons dans la première section de ce chapitre le processus de transformation des sous-ensembles inconsistants par max-résolution tel qu'il est typiquement appliqué. Dans la seconde section, nous étudions le lien entre l'ordre des opérations de max-résolution et le nombre et la taille des clauses de compensation obtenues. Nous présentons une nouvelle heuristique basée sur ces observations et discutons de son implémentation. Les résultats de l'étude expérimentale que nous avons menée sont présentés et analysés dans la troisième section. Les travaux et résultats présentés dans ce chapitre ont été précédemment publiés [AH14a, AH15a]

4.1 Transformation des sous-ensembles inconsistants par max-résolution

4.1.1 Règle de la max-résolution

La règle de la max-résolution, introduite dans la section 2.2.3, permet de transformer une formule $\Phi = \{c_i, c_j\} \cup \phi$ (où $c_i = \{x, y_1, \ldots, y_s\}$ et $c_j = \{\overline{x}, z_1, \ldots, z_t\}$ sont des clauses et ϕ est un ensemble de clauses) en une formule équivalente $\Phi' = \{cr\} \cup CC \cup \phi$ avec $cr = \{y_1, \ldots, y_s, z_1, \ldots, z_t\}$ la clause résolvante et $CC = \{cc_1, \ldots, cc_t, cc_{t+1}, \ldots, cc_{t+s}\}$ un ensemble de clauses de compensation définies comme suit :

$$
\begin{array}{ll}
cc_1 = \{x, y_1, \ldots, y_s, \overline{z}_1, z_2, \ldots, z_t\} & cc_{t+1} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_1, y_2, \ldots, y_s\} \\
cc_2 = \{x, y_1, \ldots, y_s, \overline{z}_2, \ldots, z_t\} & cc_{t+2} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_2, \ldots, y_s\} \\
\vdots & \vdots \\
cc_t = \{x, y_1, \ldots, y_s, \overline{z}_t\} & cc_{t+s} = \{\overline{x}, z_1, \ldots, z_t, \overline{y}_s\}
\end{array}
$$

On peut exprimer la taille de la résolvante cr en fonction de la taille des clauses originales c_i et c_j (en considérant que les littéraux redondants sont supprimés) :

$$
\begin{aligned}
|cr| &= |c_i \cup c_j \setminus \{x, \overline{x}\}| \\
&= |c_i| + |c_j| - |c_i \cap c_j| - 2
\end{aligned}
\tag{4.1}
$$

De la même manière, on peut exprimer le nombre de clauses de compensation (en considérant que les clauses tautologiques sont ignorées, cf. section 2.2) :

$$
|CC| = |c_i| + |c_j| - 2 * |c_i \cap c_j| - 2 \tag{4.2}
$$

et leurs tailles :

$$\begin{array}{ll} |cc_1| = |c_i| + |c_j| - |c_i \cap c_j| - 1 & |cc_{t+1}| = |c_i| + |c_j| - |c_i \cap c_j| - 1 \\ |cc_2| = |c_i| + |c_j| - |c_i \cap c_j| - 2 & |cc_{t+2}| = |c_i| + |c_j| - |c_i \cap c_j| - 2 \\ \vdots & \vdots \\ |cc_t| = |c_i| + 1 & |cc_{t+s}| = |c_j| + 1 \end{array} \tag{4.3}$$

4.1.2 Transformation des sous-ensembles inconsistants

Cette règle est utilisée pour transformer les sous-ensembles inconsistants durant le calcul de la borne inférieure. Lorsqu'un conflit (une clause vide) est détecté par la propagation unitaire (propagation unitaire simulée, SUP, ou méthode des littéraux falsifiés, FL), les solveurs construisent un sous-ensemble inconsistant ψ de la formule en analysant le graphe d'implications $G = (V, A)$ (cf. définition 19) qui décrit les étapes de propagation réalisées. Chaque clause apparaissant dans un chemin entre les sources de ce graphe (les décisions ou les clauses unitaires) et la clause unitaire appartient au sous-ensemble inconsistant. ψ peut ensuite être transformé en éliminant successivement par max-résolution les littéraux propagés participant au conflit (i.e. qui apparaissent dans le graphe d'implications). Pour une étape de max-résolution entre deux clauses $c_j, c_k \in \psi$ sur une variable $var(l_i), l_i \in V$ nous noterons $cr_{\langle var(l_i)\rangle}$ et $CC_{\langle var(l_i)\rangle}$ respectivement la clause résolvante et l'ensemble des clauses de compensation obtenues. Ces notations peuvent être étendues à toutes séquence de max-résolution $\langle var(l_{i_1}), \ldots, var(l_{i_k})\rangle$ (avec $l_{i_1}, \ldots, l_{i_k} \in V$) ssi le sous-graphe non orienté de G induit par $\{l_{i_1}, \ldots, l_{i_k}\}$ est connexe. Après une étape de max-résolution, le graphe d'implications peut être mis à jour en supprimant le littéral résolu l_i du graphe d'implications et en remplaçant les clauses c_j et c_k par la clause résolvante $cr_{\langle var(x_i)\rangle}$ générée. Cette opération est décrite formellement ci-dessous.

Définition 36 (Graphe d'implications mis à jour)**.** Soit $G = (V, A)$ un graphe d'implications. Après l'application de la max-résolution entre le prédécesseur c_j et l'unique successeur c_k d'une variable $var(l_i)$ t.q. $l_i \in V$, G peut être transformé en $G_{\langle var(l_i)\rangle} = (V_{\langle var(l_i)\rangle}, A_{\langle var(l_i)\rangle})$ avec :

$$\begin{array}{rcl} V_{\langle var(l_i)\rangle} & = & (V \setminus \{l_i\}) \setminus \{\Diamond_{c_j} \text{ si } |c_j| = 1\} \cup \{\Diamond_{cr_{\langle var(l_i)\rangle}} \text{ si } |cr_{\langle var(l_i)\rangle}| = 1\} \\ A_{\langle var(l_i)\rangle} & = & \{(l, l', c_p) \in A \text{ t.q. } l \neq l_i,\, l' \neq l_i,\, c_p \neq c_j \text{ et } c_p \neq c_k\} \cup \\ & & \{(l, l', cr_{\langle var(l_i)\rangle}) \text{ t.q. } \bar{l}, l' \in cr_{\langle var(l_i)\rangle} \text{ et } l, l' \in V_{\langle var(l_i)\rangle}\} \cup \\ & & \{(\Diamond_{cr_{\langle var(l_i)\rangle}}, l, cr_{\langle var(l_i)\rangle}) \text{ si } cr_{\langle var(l_i)\rangle} = \{l\}\} \cup \\ & & \{(l, \Box, cr_{\langle var(l_i)\rangle}) \text{ t.q. } \bar{l} \in cr_{\langle var(l_i)\rangle} \text{et } cr_{\langle var(l_i)\rangle} \text{ est falsifiée par } I\} \end{array}$$

La notion de graphe d'implications mis à jour peut être étendue à toute séquence d'étapes de max-résolution $\langle var(l_{i_1}), \ldots, var(l_{i_k})\rangle$ comme suit :

$$G_{\langle var(l_{i_1}),\ldots,var(l_{i_k})\rangle} = (\ldots((G_{\langle var(l_{i_1})\rangle})_{\langle var(l_{i_2})\rangle})\ldots)_{\langle var(l_{i_k})\rangle}$$

Propriété 5. La complexité temporelle dans le pire des cas de la mise à jour d'un graphe d'implications après une opération de max-résolution entre deux clauses c_j et c_k sur une variable x_i est en $\mathcal{O}(|c_j| + |c_k|)$.

Démonstration. La mise à jour du graphe d'implications après une opération de max-résolution entre deux clauses c_j et c_k sur une variable x_i implique la suppression du nœud correspondant à la variable x_i, ce qui peut être fait en temps constant, et le remplacement des arcs étiquetés par les clauses c_j et c_k par des arcs étiquetés par la clause résolvante obtenue. Comme le nombre de ces arcs est borné par le nombre de littéraux des clauses c_j et c_k, la complexité temporelle de l'application de la mise à jour est bien en $\mathcal{O}(|c_j| + |c_k|)$. □

Quand tous les littéraux du graphe d'implications ont été éliminés, la dernière clause résolvante ne contient plus de littéraux propagés. Elle est soit vide soit directement falsifiée par les décisions. Le conflit est alors mémorisé et la propagation unitaire n'est plus nécessaire à sa détection, ni au nœud courant ni dans les sous-nœuds de l'arbre de recherche.

On peut noter que si un littéral propagé $l_i \in V$ peut avoir une seule source de propagation dans un graphe d'implications, il peut réduire plusieurs autres clauses et donc mener à la propagation de plusieurs autres littéraux. Dans cette situation, la max-résolution ne peut être appliquée sur $var(l_i)$ que lorsque les clauses qu'il réduit ont toutes été transformées par max-résolution en un unique résolvant intermédiaire. Pour cette raison, les étapes de max-résolution sont généralement appliquées dans l'ordre inverse des propagations (*reverse propagation order*, RPO) puisque le dernier littéral propagé du graphe d'implications ne peut réduire que la clause vide. L'algorithme 4.1 montre cette manière d'appliquer la max-résolution sur un sous-ensemble inconsistant. Tant que le graphe d'implications n'est pas vide, il choisit le littéral propagé le plus récemment et détermine sa source de propagation et la clause qu'il réduit (lignes 3-5). Puis, la max-résolution est appliquée (ligne 6) et le graphe d'implications est mis à jour (ligne 7). L'exemple suivant illustre cette méthode de transformation.

Algorithme 4.1 : Fonction `transformation_sous_ensemble_inconsistant` par max-résolution appliquée dans l'ordre inverse des propagations (heuristique RPO).

Data : Une formule Φ, une interprétation I qui falsifie une clause $c_f \in \Phi$ et un graphe d'implications $G = (V, A)$ décrivant les étapes de propagation unitaire ayant conduit à falsifier c_f.
Result : La formule Φ transformée.

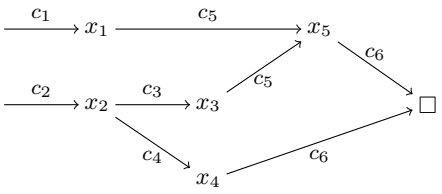
```
1 begin
2     while |V| > 0 do
3         l ← le littéral de V propagé le plus récemment;
4         c1 ← red_G(l);
5         c2 ← src_G(l);
6         Φ ← max_resolution(Φ,c1,var(l),c2);
7         G ← G_⟨var(l)⟩;
```

Propriété 6. Pour un sous-ensemble inconsistant ψ en entrée composé de s clauses de taille inférieure ou égale à k, la complexité temporelle dans le pire des cas de l'algorithme 4.1 est en $\mathcal{O}(s * k^2)$.

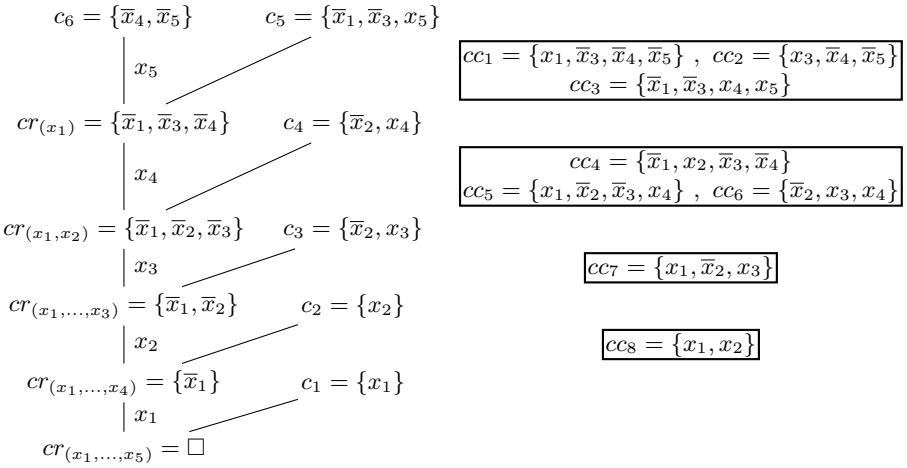
Démonstration. Nous avons vu que la complexité temporelle de l'application de la règle de la max-résolution entre deux clauses c_1 et c_2 est en $\mathcal{O}(|c_1| * |c_2|)$ (cf. propriété 2). De

plus, la mise à jour du graphe d'application après une opération de max-résolution peut être réalisée en temps linéaire par rapport à la taille des clauses résolues (cf. propriété 5). Comme il faut $s-1$ opérations de max-résolution pour transformer le sous-ensemble inconsistant ψ de taille s en entrée, la complexité temporelle de l'algorithme est bien en $\mathcal{O}(s * k^2)$ avec k est la taille maximale des clauses de ψ. □

Exemple 15. Considérons une formule non-valuée $\Phi = \{c_1, c_2, \ldots, c_6\}$ définie sur un ensemble de variables $X = \{x_1, \ldots, x_5\}$ avec $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x}_2, x_3\}$, $c_4 = \{\overline{x}_2, x_4\}$, $c_5 = \{\overline{x}_1, \overline{x}_3, x_5\}$ et $c_6 = \{\overline{x}_4, \overline{x}_5\}$. L'application de la propagation unitaire simulée sur Φ conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5\rangle$ (x_1 est propagée par c_1, puis x_2 par c_2, etc.). La clause c_6 est falsifiée. La figure 4.1a montre le graphe d'implications correspondant tandis que la figure 4.1b montre les étapes de max-résolution appliquées sur la formule Φ par l'algorithme 4.1, avec les clauses de compensation encadrées. Les clauses originales $c_1, \ldots, c_6$ sont supprimées de la formule tandis que la clause résolvante finale $cr_{(x_1,\ldots,x_5)} = \square$ et les clauses de compensation $cc_1 = \{x_1, \overline{x}_3, \overline{x}_4, \overline{x}_5\}$, $cc_2 = \{x_3, \overline{x}_4, \overline{x}_5\}$, $cc_3 = \{\overline{x}_1, \overline{x}_3, x_4, x_5\}$, $cc_4 = \{\overline{x}_1, x_2, \overline{x}_3, \overline{x}_4\}$, $cc_5 = \{x_1, \overline{x}_2, \overline{x}_3, x_4\}$, $cc_6 = \{\overline{x}_2, x_3, x_4\}$, $cc_7 = \{x_1, \overline{x}_2, x_3\}$ et $cc_8 = \{x_1, x_2\}$ sont ajoutées. On obtient la formule $\Phi' = \{\square, cc_1, \ldots, cc_8\}$. On peut noter que huit clauses de compensation ont été ajoutées, dont une de taille deux, trois de taille trois et quatre de taille quatre.



(a) Graphe d'implications.



(b) Étapes de max-résolution.

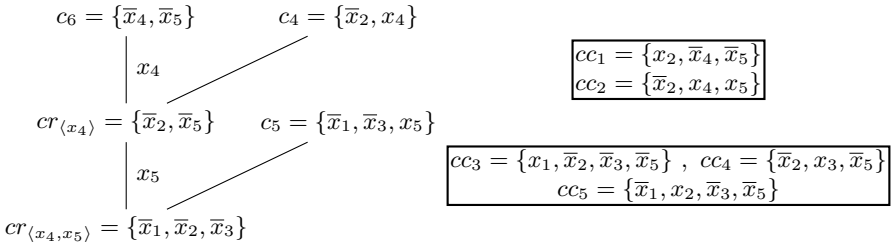
FIGURE 4.1 – Graphe d'implications et étapes de max-résolution appliquées sur la formule Φ dans l'exemple 15.

4.2 Réduire le nombre et la taille des clauses de compensation

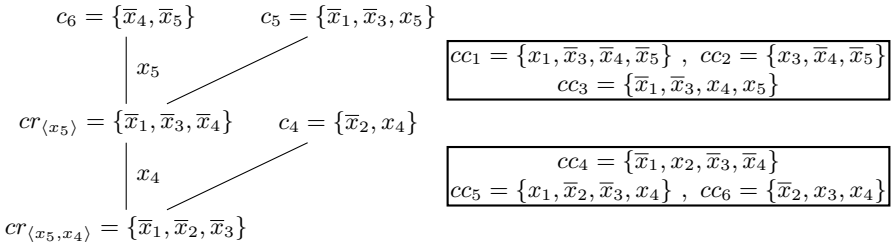
Nous avons vu dans la section précédente comment les sous-ensembles inconsistants pouvaient être transformés grâce à la règle de la max-résolution. Ces transformations génèrent des clauses de compensation, et donc la taille de la formule peut croître rapidement ce qui empêche une utilisation généralisée de cette méthode de transformation. Nous nous intéressons dans cette section à l'impact de l'ordre d'application des étapes de max-résolution sur le nombre et la taille des clauses de compensation. Nous étudions dans la première partie de cette section ce lien de manière théorique et nous présentons dans la seconde partie une nouvelle heuristique permettant de définir l'ordre d'application des étapes de max-résolution.

4.2.1 Impact de l'ordre des étapes de max-résolution sur les clauses de compensation

Il est intéressant d'observer que le nombre et les tailles des clauses de compensation ajoutées à la formule dépendent de l'ordre d'application des étapes de max-résolution. Considérons par exemple le sous-ensemble $\{c_4, c_5, c_6\}$ de la formule Φ de l'exemple 15. Deux séquences d'étapes de max-résolution sont possibles : $\langle x_4, x_5 \rangle$ et $\langle x_5, x_4 \rangle$. La figure 4.2 montre les transformations obtenues par l'application de ces deux séquences. On obtient dans les deux cas le même résolvant final $cr_{\langle x_4,x_5 \rangle} = cr_{\langle x_5,x_4 \rangle} = \{\overline{x}_1, \overline{x}_2, \overline{x}_3\}$ mais dans le second cas six clauses de compensation sont générées (deux de taille trois et quatre de taille quatre) tandis que dans le premier cas seulement cinq clauses sont ajoutées (trois de taille trois et deux de taille quatre).



(a) Selon la séquence $\langle x_4, x_5 \rangle$



(b) Selon la séquence $\langle x_5, x_4 \rangle$

FIGURE 4.2 – Application de la max-résolution sur l'ensemble de clauses $\{c_4, c_5, c_6\}$.

Il y a une relation directe entre la taille des clauses résolvantes obtenues et le nombre et la taille des clauses de compensation ajoutées à la formule. Comme nous l'avons vu dans la section précédente, plus les clauses résolues sont de grande taille plus la taille du résolvant, le nombre de clauses de compensation et leurs tailles sont grands. Par conséquent, en faisant d'abord les opérations de max-résolution qui produisent de petits résolvants intermédiaires on peut réduire le nombre et la taille des clauses de compensation produites par les étapes de max-résolution suivantes qui utiliserons ces résolvants.

Pour évaluer quelles étapes de résolution doivent être appliquées en premier, nous attribuons un score aux littéraux apparaissant dans le graphe d'implications. Ce score dépend de la taille du résolvant obtenu si l'on applique la max-résolution sur la variable correspondante. Lorsqu'un littéral a plusieurs successeurs (i.e. il réduit plus d'une clause du sous-ensemble inconsistant) alors, comme nous l'avons expliqué dans la section précédente, on ne peut pas appliquer directement la max-résolution et on donne un score infini au littéral. Formellement, pour un littéral l_i apparaissant dans un graphe d'implications G :

$$score(l_i) = \begin{cases} |src_G(l_i)| + |c| - |src_G(l_i) \cap c| - 2 & \text{si } red_G(l_i) = \{c\} \\ \infty & \text{si } |red_G(l_i)| > 1 \end{cases} \tag{4.4}$$

On peut montrer que pour deux littéraux adjacents dans un graphe d'implications, appliquer la max-résolution en premier sur la variable de celui de plus petit score produit toujours un plus petit nombre de clauses de compensation.

Propriété 7. Soit $G = (V, A)$ un graphe d'implications. Considérons deux littéraux l_i et l_j (avec $i \neq j$) tels que $l_i, l_j \in V$ et (l_i, l_j) ou $(l_j, l_i) \in A$. Nous avons la relation suivante :

$$score(l_i) \leq score(l_j) \text{ si et seulement si } |CC_{\langle var(x_i), var(x_j)\rangle}| \leq |CC_{\langle var(x_j), var(x_i)\rangle}|$$

Démonstration. On suppose sans perte de généralité que les sources de propagation de l_i et l_j sont respectivement c_{k_1} et c_{k_2} et que les clauses réduites par l_i et l_j sont respectivement c_{k_2} et c_{k_3}. Nous allons exprimer $|CC_{\langle var(l_i), var(l_j)\rangle}|$ et $|CC_{\langle var(l_j), var(l_i)\rangle}|$ en fonction des tailles de c_{k_1}, c_{k_2} et c_{k_3}. $|CC_{\langle var(l_i), var(l_j)\rangle}|$ est le nombre de clauses de compensation obtenues en appliquant la max-résolution entre c_{k_1} et c_{k_2} sur $var(l_i)$ puis entre $c_{\langle var(l_i)\rangle}$ (le résolvant obtenu lors de l'étape précédente) et c_{k_3} sur $var(l_j)$. Ces deux étapes de résolution produisent respectivement les ensembles de clauses de compensation $CC_{\langle var(l_i)\rangle}$ et $CC_{\langle var(l_j)\rangle|\langle var(l_i)\rangle}$ (on note $CC_{\langle x_l\rangle|\langle x_{m_1},\ldots,x_{m_p}\rangle}$ l'ensemble des clauses de compensation obtenues en appliquant la max-résolution sur une variable x_l après la séquence d'étapes de résolution $\langle x_{m_1}, \ldots, x_{m_p}\rangle$). On a :

$$|CC_{\langle var(l_i), var(l_j)\rangle}| = |CC_{\langle var(l_i)\rangle}| + |CC_{\langle var(l_j)\rangle|\langle var(l_i)\rangle}| \tag{4.5}$$

D'après la définition 4.2, on sait que :

$$|CC_{\langle var(l_i)\rangle}| = |c_{k_1}| + |c_{k_2}| - 2 * |c_{k_1} \cap c_{k_2}| - 2 \tag{4.6}$$

et

$$|CC_{\langle var(l_j)\rangle|\langle var(l_i)\rangle}| = |cr_{\langle var(l_i)\rangle}| + |c_{k_3}| - 2 * |cr_{\langle var(l_i)\rangle} \cap c_{k_3}| - 2 \tag{4.7}$$

De plus, d'après la définition de la règle de la max-résolution (cf. section 2.2.3) nous savons que $cr_{\langle var(l_i)\rangle} = c_{k_1} \cup c_{k_2} \setminus \{x_i, \overline{x}_i\}$. Comme $c_{k_3} \cap \{x_i, \overline{x}_i\} = \emptyset$, on peut déduire :

$$|cr_{\langle var(l_i)\rangle} \cap c_{k_3}| = |c_{k_1} \cap c_{k_3}| + |c_{k_2} \cap c_{k_3}| - |c_{k_1} \cap c_{k_2} \cap c_{k_3}| \tag{4.8}$$

Nous avons donc par 4.7, 4.1 et 4.8 :

$$|CC_{\langle var(l_j)\rangle|\langle var(l_i)\rangle}| = |c_{k_1}| + |c_{k_2}| - |c_{k_1} \cap c_{k_2}| - 2 + |c_{k_3}| \\ - 2 * (|c_{k_1} \cap c_{k_3}| + |c_{k_2} \cap c_{k_3}| - |c_{k_1} \cap c_{k_2} \cap c_{k_3}|) \quad (4.9)$$

On obtient donc par 4.5, 4.6 et 4.9 puis 4.4 :

$$\begin{aligned} |CC_{\langle var(l_i),var(l_j)\rangle}| &= 2 * |c_{k_1}| + 2 * |c_{k_2}| + |c_{k_3}| - 3 * |c_{k_1} \cap c_{k_2}| - 2 * |c_{k_1} \cap c_{k_3}| \\ &\quad - 2 * |c_{k_2} \cap c_{k_3}| + 2 * |c_{k_1} \cap c_{k_2} \cap c_{k_3}| - 6 \quad (4.10) \\ &= score(l_i) + k \end{aligned}$$

avec $k = |c_{k_1}|+|c_{k_2}|+|c_{k_3}|-2*|c_{k_1}\cap c_{k_2}|-2*|c_{k_1}\cap c_{k_3}|-2*|c_{k_2}\cap c_{k_3}|+2*|c_{k_1}\cap c_{k_2}\cap c_{k_3}|-4$.

De la même manière, on peut obtenir :

$$\begin{aligned} |CC_{\langle var(l_j),var(l_i)\rangle}| &= |c_{k_1}| + 2 * |c_{k_2}| + 2 * |c_{k_3}| - 2 * |c_{k_1} \cap c_{k_2}| - 2 * |c_{k_1} \cap c_{k_3}| \\ &\quad - 3 * |c_{k_2} \cap c_{k_3}| + 2 * |c_{k_1} \cap c_{k_2} \cap c_{k_3}| - 6 \\ &= score(l_j) + k \end{aligned} \quad (4.11)$$

On a donc bien $score(l_i) \leq score(l_j)$ ssi $|CC_{\langle var(l_i),var(l_j)\rangle}| \leq |CC_{\langle var(l_j),var(l_i)\rangle}|$. □

La propriété suivante établie un lien, pour deux littéraux adjacents dans un graphe d'implications, entre le score des littéraux et la taille maximale des clauses de compensation produites.

Propriété 8. Soit $G = (V, A)$ un graphe d'implications. Considérons deux littéraux l_i et l_j (avec $i \neq j$) tels que $l_i, l_j \in V$ et (l_i, l_j) ou $(l_j, l_i) \in A$. Nous avons la relation suivante : $score(l_i) \leq score(l_j)$ si et seulement si la taille maximale des clauses de compensation obtenues en appliquant la séquence d'étapes de max-résolution $\langle var(l_i), var(l_j)\rangle$ est inférieure ou égale à celle obtenue avec la séquence $\langle var(l_j), var(l_i)\rangle$.

Démonstration. Comme pour la preuve de la propriété 7, on suppose sans perte de généralité que les sources de propagation de l_i et l_j sont respectivement c_{k_1} et c_{k_2} et que les clauses réduites par l_i et l_j sont respectivement c_{k_2} et c_{k_3}. Nous avons vu (cf. équation 4.3) que la taille des clauses de compensation obtenues en appliquant la max-résolution entre c_{k_1} et c_{k_2} sur la variable $var(l_i)$ varie de $|c_{k_1}| + 1$ à $|c_{k_1}| + |c_{k_2}| - |c_{k_1} \cap c_{k_2}| - 1$ et de $|c_{k_2}| + 1$ à $|c_{k_1}| + |c_{k_2}| - |c_{k_1} \cap c_{k_2}| - 1$. La taille des clauses de compensation obtenue par la seconde étape de max-résolution entre c_{k_3} et $cr_{\langle var(l_i)\rangle}$ varie de $|cr_{\langle var(l_i)\rangle}| + 1$ à $|cr_{\langle var(l_i)\rangle}|+|c_{k_3}|-|cr_{\langle var(l_i)\rangle}\cap c_{k_3}|-1$ et de $|c_{k_3}|+1$ à $|cr_{\langle var(l_i)\rangle}|+|c_{k_3}|-|cr_{\langle var(l_i)\rangle}\cap c_{k_3}|-1$. Nous notons respectivement $ub_{\langle var(l_i)\rangle}$ et $ub_{\langle var(l_i),var(l_j)\rangle}$ les bornes supérieures de ces intervalles, que l'on peut exprimer en fonction des tailles de c_{k_1}, c_{k_2} et c_{k_3} comme suit :

$$\begin{aligned} ub_{\langle var(l_i)\rangle} &= |c_{k_1}| + |c_{k_2}| - |c_{k_1} \cap c_{k_2}| - 1 \\ &= score(l_i) + 1 \end{aligned} \quad (4.12)$$

et

$$\begin{aligned} ub_{\langle var(l_i),var(l_j)\rangle} &= |cr_{\langle var(l_i)\rangle}| + |c_{k_3}| - |cr_{\langle var(l_i)\rangle} \cap c_{k_3}| - 1 \\ &= score(l_i) + |c_{k_3}| - |cr_{\langle var(l_i)\rangle} \cap c_{k_3}| - 1 \\ &= score(l_i) + |c_{k_3}| - |c_{k_1} \cap c_{k_3}| - |c_{k_2} \cap c_{k_3}| + |c_{k_1} \cap c_{k_2} \cap c_{k_3}| - 1 \\ &= score(l_i) + score(l_j) - |c_{k_2}| - |c_{k_1} \cap c_{k_3}| + |c_{k_1} \cap c_{k_2} \cap c_{k_3}| + 1 \end{aligned} \quad (4.13)$$

De la même manière, on peut exprimer en fonction des tailles de c_{k_1}, c_{k_2} et c_{k_3} les bornes supérieures des intervalles des tailles des clauses de compensation obtenues en appliquant la max-résolution sur $var(l_j)$ puis sur $var(l_i)$:

$$\begin{aligned} ub_{\langle var(l_j)\rangle} &= |c_{k_2}| + |c_{k_3}| - |c_{k_2} \cap c_{k_3}| - 1 \\ &= score(l_j) + 1 \end{aligned} \tag{4.14}$$

et

$$\begin{aligned} ub_{\langle var(l_j), var(l_i)\rangle} &= |cr_{\langle var(l_j)\rangle}| + |c_{k_1}| - |cr_{\langle var(l_j)\rangle} \cap c_{k_1}| - 1 \\ &= score(l_j) + |c_{k_1}| - |cr_{\langle var(l_j)\rangle} \cap c_{k_1}| - 1 \\ &= score(l_j) + |c_{k_1}| - |c_{k_1} \cap c_{k_2}| - |c_{k_1} \cap c_{k_3}| + |c_{k_1} \cap c_{k_2} \cap c_{k_3}| - 1 \\ &= score(l_j) + score(x_i) - |c_{k_2}| - |c_{k_1} \cap c_{k_3}| + |c_{k_1} \cap c_{k_2} \cap c_{k_3}| + 1 \end{aligned} \tag{4.15}$$

Donc si $score(l_i) \leq score(l_j)$ on a $ub_{\langle var(l_i)\rangle} \leq ub_{\langle var(l_j)\rangle}$ et inversement. Les secondes bornes supérieures ($ub_{\langle var(l_i), var(l_j)\rangle}$ et $ub_{\langle var(l_j), var(l_i)\rangle}$) sont égales quels que soient les scores de x_i et x_j. □

4.2.2 Heuristique SIR

Nous avons montré dans les propriétés 7 et 8 que pour deux littéraux voisins dans un graphe d'implications, appliquer la max-résolution d'abord sur celui de plus petit score (i.e. celui qui produira la plus petite clause résolvante) permet de produire moins de clauses de compensation et de réduire leurs tailles. Nous proposons donc une heuristique appliquant les étapes de max-résolution par ordre croissant de score des littéraux (algorithme 4.2). Cette heuristique calcule d'abord les scores des littéraux apparaissant dans le graphe d'implications (ligne 2). Puis elle choisie le littéral l de plus petit score (ligne 4) et sélectionne la clause c_1 ayant causé sa propagation et la clause c_2 qu'il réduit (lignes 5-6). Puis elle applique la max-résolution entre c_1 et c_2 sur $var(l)$ et actualise le graphe

Algorithme 4.2 : Fonction `transformation_sous_ensemble_inconsistant` par max-résolution appliquée dans l'ordre des scores croissants (heuristique SIR).

Data : Une formule Φ, une interprétation I qui falsifie une clause $c_f \in \Phi$ et un graphe d'implications $G = (V, A)$ décrivant les étapes de propagation unitaire ayant conduit à falsifier c_f.
Result : La formule Φ transformée.

```
1 begin
2     for l ∈ V do calcul du score de l ;
3     while |V| > 0 do
4         l ← le littéral de V de plus petit score;
5         c1 ← red_G(l);
6         c2 ← src_G(l);
7         Φ ← max_resolution(Φ,c1,var(l),c2);
8         G ← G_⟨var(l)⟩;
9         for l' ∈ c1 ∪ c2 \ {l} do calcul du score de l' ;
```

d'implications avant de mettre à jour les scores des littéraux apparaissant dans c_1 et c_2. Ces étapes sont répétées jusqu'à ce que tous les littéraux apparaissant dans les clauses du sous-ensemble inconsistant aient été traités. Nous appellerons cette heuristique SIR (*smallest intermediary resolvent*) dans la suite de ce chapitre.

Propriété 9. Pour un sous-ensemble inconsistant ψ en entrée composé de s clauses de taille inférieure ou égale à k, la complexité temporelle dans le pire des cas de l'algorithme 4.2 est en $\mathcal{O}(s * k^2)$.

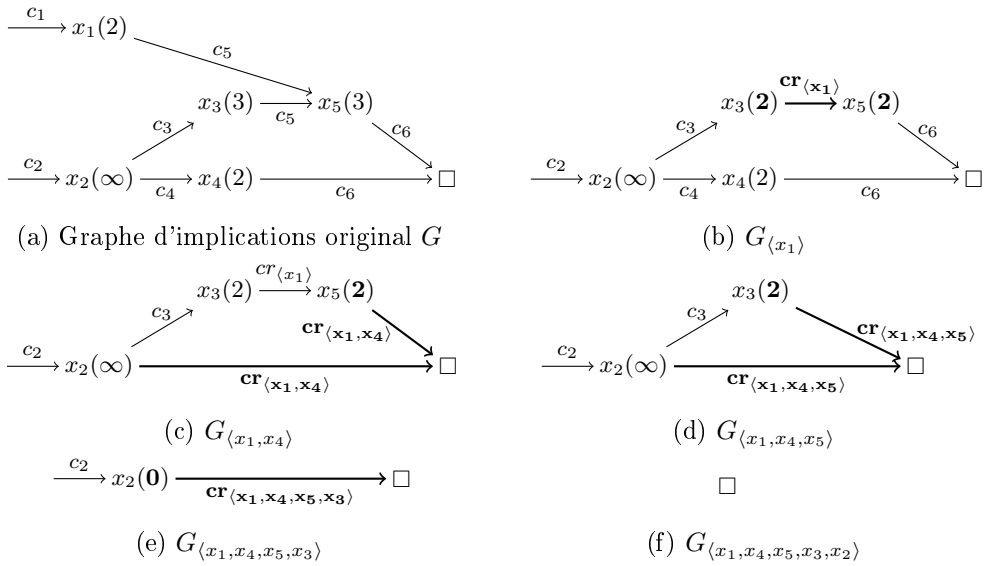
Démonstration. Nous avons vu que la complexité temporelle dans le pire des cas de la transformation par max-résolution d'un sous-ensemble inconsistant ψ dans l'ordre inverse des propagation (heuristique RPO) est en $\mathcal{O}(s * k^2)$ avec s la taille de ψ et k la taille maximale de ses clauses (cf. algorithme 4.1 et propriété 6). Les seules différences entre l'heuristique SIR présentée dans l'algorithme 4.2 et l'heuristique RPO est le calcul des scores des littéraux. Pour un littéral l_i donné, ce score peut être calculé en temps linéaire par rapport à la taille de la source de propagation de l_i et de la clause qu'il réduit. Sa complexité temporelle dans le pire des cas est donc en $\mathcal{O}(k)$ si k est la taille maximale des clauses du sous-ensemble inconsistant. Ce calcul est réalisé une fois pour chaque littéral du sous-ensemble inconsistant au début de l'algorithme avec une complexité temporelle en $\mathcal{O}(s*k)$ (avec s la taille de ψ). Puis, les scores des littéraux apparaissant dans les clauses résolues sont actualisés après chaque opération de max-résolution. Dans cette dernière situation, au plus $2 * k$ littéraux voient leur score recalculé. Comme $s - 1$ opérations de max-résolution sont réalisées, la complexité temporelle dans le pire des cas de la mise à jour des scores des littéraux est en $\mathcal{O}(s * k^2)$. La complexité temporelle de l'algorithme dans son ensemble est donc également en $\mathcal{O}(s * k^2)$. □

L'exemple suivant illustre le fonctionnement de l'heuristique SIR.

Exemple 16. Considérons à nouveau la formule Φ de l'exemple 15. Nous avons vu que l'application de la propagation unitaire simulée sur Φ produisait la séquence de propagations $\langle x_1@c_1, x_2@c_2, \ldots, x_5@c_5\rangle$ et que la clause c_6 était falsifiée. La figure 4.3 montre l'évolution du graphe d'implications pendant la transformation, avec les scores des variables entre parenthèses. Les étapes de max-résolution appliquées sont présentées dans la figure 4.4.

Il y a initialement deux littéraux de plus faible score 2, x_1 et x_4. L'algorithme applique la max-résolution entre la clause causant la propagation de x_1, c_1, et la clause qu'elle réduit c_5. Le résolvant intermédiaire $cr_{\langle x_1\rangle} = \{\overline{x}_3, x_5\}$ est produit. L'algorithme met à jour le graphe d'implications en supprimant le nœud x_1, en remplaçant l'étiquette c_5 de l'arc (x_3, x_5) par $cr_{\langle x_1\rangle}$ et en actualisant les scores des variables x_3 et x_5. La figure 4.3b montre le graphe d'implications ainsi obtenu, avec en gras les arcs modifiés et les scores mis à jour.

Puis l'algorithme prend la variable suivante de score 2 (x_4) et applique la max-résolution entre sa source de propagation (c_4) et la clause qu'elle réduit (c_6). Il met à jour le graphe d'implications (figure 4.3c). Le même traitement est appliqué successivement sur les variables x_5 (figure 4.3d) puis x_3 (figure 4.3e). À ce stade, les deux clauses réduites par x_2 ont été fusionnées et l'on peut donc appliquer la max-résolution entre sa source de propagation c_2 et le résolvant intermédiaire $cr_{\langle x_1,x_4,x_5,x_3\rangle}$. Cette dernière étape de max-résolution produit un résolvant vide et, après avoir été mis à jour, le graphe d'implications ne contient plus aucune variable (figure 4.3f).



(a) Graphe d'implications original G (b) $G_{\langle x_1 \rangle}$

(c) $G_{\langle x_1, x_4 \rangle}$ (d) $G_{\langle x_1, x_4, x_5 \rangle}$

(e) $G_{\langle x_1, x_4, x_5, x_3 \rangle}$ (f) $G_{\langle x_1, x_4, x_5, x_3, x_2 \rangle}$

FIGURE 4.3 – Évolution du graphe d'implications pendant la transformation de la formule Φ de l'exemple 16.

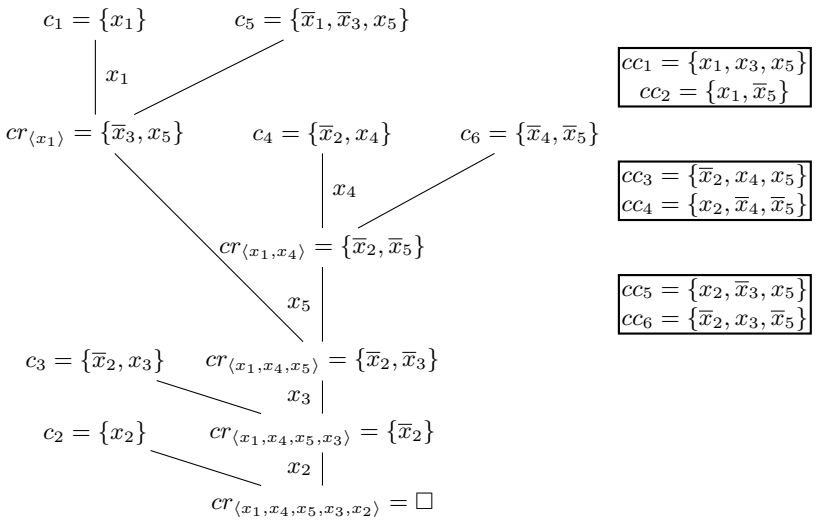


FIGURE 4.4 – Étapes de max-résolution appliquées sur la formule Φ dans l'exemple 16.

On obtient la formule transformée $\Phi'' = \{\square, cc_1, \ldots, cc_6\}$ qui contient, en plus de la clause résolvante vide $\square$, six clauses de compensation (cinq de taille trois et une de taille deux). On peut remarquer que cette transformation produit deux clauses de moins qu'avec l'ordre inverse des propagations utilisé habituellement (cf. exemple 15) et que les clauses obtenues sont plus courtes.

4.3 Étude expérimentale

Nous avons implémenté les deux ordres d'application de la max-résolution dans AHMAXSAT. Nous appelons la variante utilisant l'ordre inverse des propagations (*reverse propagation order*, RPO) AHMAXSAT^{RPO} et la variante utilisant notre heuristique (*smallest intermediary resolvant heuristic*, SIR) AHMAXSAT^{SIR}. Nous avons testé ces deux variantes suivant le protocole expérimental décrit dans la section 2.6.

Les résultats sont présentés dans la table 4.1. Les colonnes S, D et T donnent respectivement le nombre d'instances résolues, le nombre moyen de nœuds de l'arbre de recherche explorés et le temps moyen de résolution. On peut remarquer qu'AHMAXSAT^{SIR} résout 34 instances de plus qu'AHMAXSAT^{RPO}. Le temps moyen de résolution, mesuré sur les instances résolues par les deux variantes du solveur, est réduit significativement par l'heuristique SIR (-22,8% en moyenne). Le gain est particulièrement régulier sur les instances aléatoires, où il varie entre -10% et -32,7%. Les résultats sont plus disparates sur les instances crafted. La réduction du temps moyen de résolution est parfois importante (e.g. pms/crafted/min-enc/kbtree, -59,5% ou wpms/crafted/random-net, -41,2%)) mais sur d'autres classes le gain est faible (e.g. pms/crafted/frb, -1,6% ou pms/crafted/maxclique, -0,6%). Dans certains cas, on observe même une augmentation du temps moyen de résolution (e.g. wpms/crafted/CSG, +3,8% ou wpms/crafted/ramsey, +3%). On peut aussi remarquer que le nombre moyen de nœuds explorés par AHMAXSAT^{SIR} est sensiblement plus faible que celui d'AHMAXSAT^{RPO} (-14,5%, colonnes D). Là encore, les résultats sont plus réguliers sur les instances aléatoires (de -10% à -35,3%) que sur les instances crafted (de +60,8% à -55,7%). Les figures 4.5a et 4.5b comparent respectivement, instance par instance, le temps moyen de résolution et le nombre de nœuds de l'arbre de recherche explorés d'AHMAXSAT^{SIR} et d'AHMAXSAT^{RPO}. On peut voir dans les deux cas que le gain est réparti de manière relativement uniforme sur l'ensemble des instances du benchmark.

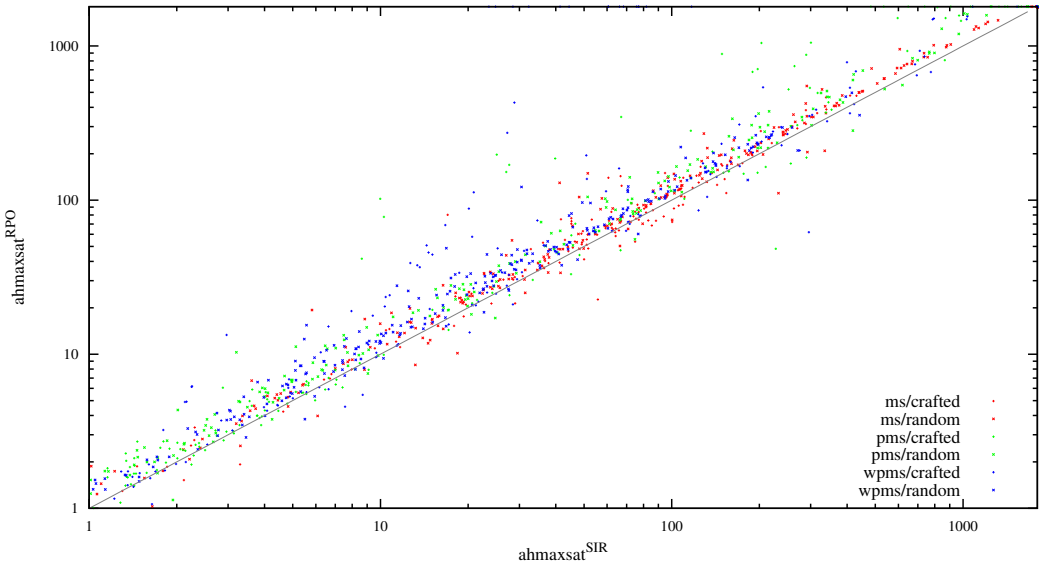
Essayons d'analyser l'impact de l'heuristique SIR sur le comportement d'AHMAXSAT. La table 4.2 présente des statistiques détaillées du fonctionnement interne du solveur. On peut tout d'abord noter que les moyennes du nombre et de la taille des clauses de compensation générées lors de la transformation des sous-ensembles inconsistants sont réduites par l'heuristique SIR de respectivement -30% et -20% (colonnes NB et SZ). On pourrait s'attendre à ce qu'AHMAXSAT^{SIR} explore plus rapidement l'espace de recherche et donc à ce que le nombre de nœuds explorés par seconde (colonnes D/s) soit sensiblement plus élevé. Si c'est le cas sur certaines classes d'instances (e.g. pms/crafted/reversi, +46,2% ou wpms/random/wpmax2sar, +29%), sur d'autres à l'inverse l'exploration de l'arbre de recherche est plus lente avec l'heuristique SIR. La moyenne sur l'ensemble du benchmark est sensiblement équivalente pour les deux variantes du solveur. Ceci peut s'expliquer par les différences entre les traitements effectués lors du calcul de la borne inférieure. En effet, sur la plupart des classes d'instances on peut voir qu'AHMAXSAT^{SIR} fait davantage d'opérations de propagation unitaire à chaque nœud de l'arbre de recherche qu'AHMAXSAT^{RPO} (en moyenne +6%, colonnes P/D). En conséquence, il détecte davantage de sous-ensembles inconsistants (+9%, colonnes □/D). Donc l'heuristique SIR permet d'améliorer la qualité de la borne inférieure, ce qui explique la réduction du nombre de nœuds explorés par le solveur (table 4.1, colonnes D). Mais ce traitement supplémentaire a un coût en termes de temps de calcul qui affecte la vitesse d'exploration de l'arbre de recherche. Cela réduit donc, voire annule, le gain en vitesse dû à la réduction du nombre

TABLE 4.1 – Comparaison des ordre d'application des étapes de max-résolution RPO et SIR. Les deux premières colonnes montrent les classes d'instances et le nombre d'instances par classe. Pour chaque variante d'AHMAXSAT, les colonnes S, D et T donnent respectivement le nombre d'instances résolues, le nombre moyens de nœuds explorés et le temps moyen d'exécution. Les colonnes marquées avec une étoile ne prennent en compte que les instances résolues par toutes les variantes.

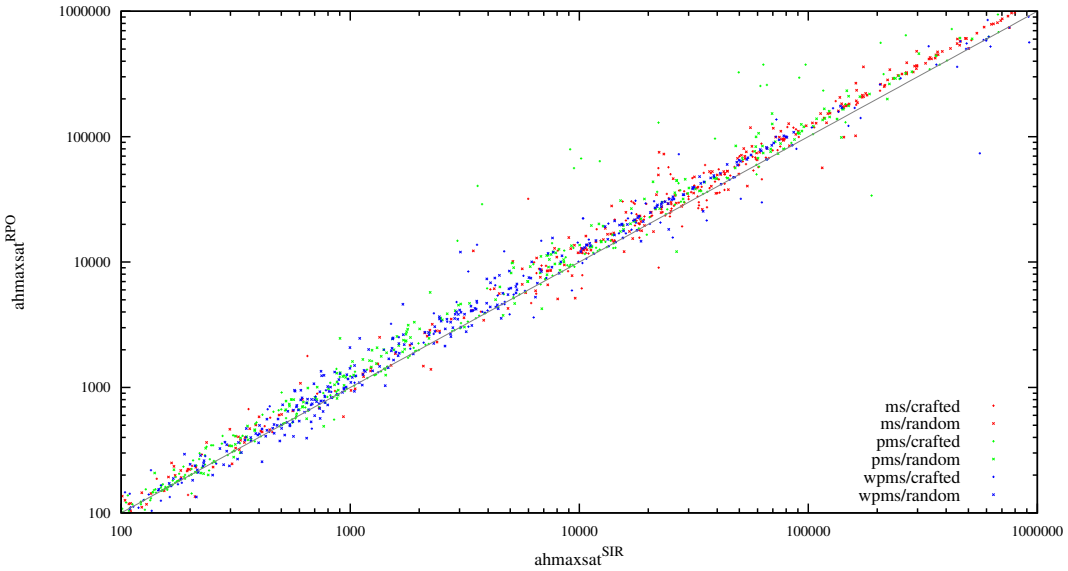
instances classes			#	AHMAXSAT^{SIR}			AHMAXSAT^{RPO}			Δ_{D*}	Δ_{T*}
				S	D*	T*	S	D*	T*		
ms	crafted	bipartite	100	100	34915	**90,7**	100	40608	101,1	-14%	-10,2%
		maxcut	67	56	171380	**44,7**	56	208687	51,1	-17,9%	-12,5%
		set-covering	10	0	-	-	0	-	-	-	-
	random	highgirth	82	**7**	4650016	**1072,9**	6	5260114	1238	-11,6%	-13,3%
		max2sat	100	100	45444	**89,5**	100	60277	115,4	-24,6%	-22,4%
		max3sat	100	100	340968	**231,2**	100	424779	265,9	-19,7%	-13,1%
		min2sat	96	96	1046	**2,6**	96	1188	2,9	-12%	-11,5%
pms	crafted	frb	25	5	585949	**154,2**	5	609626	156,6	-3,9%	-1,6%
		job-shop	3	0	-	-	0	-	-	-	-
		maxclique	158	**133**	19493	**18,7**	132	18686	18,8	4,3%	-0,6%
		maxone	140	**109**	132871	**37,4**	108	130947	44,8	1,5%	-16,4%
		min-enc/kbtree	42	**34**	113696	**242,9**	24	256720	599,5	-55,7%	-59,5%
		pseudo/miplib	4	2	288	$< 0,1$	2	271	$< 0,1$	6,3%	0%
		reversi	44	**8**	4067	**55**	7	6612	147,3	-38,5%	-62,6%
		scheduling	5	0	-	-	0	-	-	-	-
	random	min2sat	60	**58**	7955	**113,7**	55	12297	168,8	-35,3%	-32,7%
		min3sat	60	58	91569	**249,9**	58	115747	311,6	-20,9%	-19,8%
		pmax2sat	60	60	1021	**3,2**	60	1349	4,4	-24,3%	-27%
		pmax3sat/hi	30	30	84944	**60,1**	30	99455	67,2	-14,6%	-10,6%
wpms	crafted	auction	40	40	310021	**115,5**	40	300832	141,3	3,1%	-18,3%
		CSG	10	4	54813	452,3	4	54569	**435,8**	0,4%	3,8%
		frb	34	14	212369	**58,7**	14	220618	64,3	-3,7%	-8,6%
		min-enc	74	**64**	20057	**86,2**	63	12471	125,5	60,8%	-31,3%
		pseudo/miplib	12	4	1505	**158,8**	4	2290	306,4	-34,3%	-48,2%
		ramsey	15	4	157559	44,8	4	160686	**43,5**	-1,9%	3%
		random-net	32	**3**	391871	**318,5**	2	823476	541,2	-52,4%	-41,2%
		set-covering	45	**25**	3743	**28,1**	10	5788	136,6	-35,3%	-79,4%
		wmaxcut	48	46	26985	**49,4**	46	31788	54,4	-15,1%	-9,1%
	random	wmax2sat	120	120	3993	**45,4**	120	4794	57,3	-16,7%	-20,7%
		wmax3sat	40	40	36836	**99,5**	40	46786	117,3	-21,3%	-15,1%
		wpmax2sat	90	90	575	**5,5**	90	639	7,5	-10%	-26,4%
		wpmax3sat/hi	30	30	30451	**84,1**	30	37451	103,6	-18,7%	-18,8%
Ensemble			1776	**1440**	94717	**82,3**	1406	110825	106,6	-14,5%	-22,8%

TABLE 4.2 – Statistiques détaillées d'AHMAXSAT^{SIR} et d'AHMAXSAT^{RPO}. La première colonne donne les classes d'instances. Pour chaque variantes, les colonnes NB et SZ donnent respectivement le nombre moyen et la taille moyenne des clauses de compensation générées lors de la transformation des sous-ensembles inconsistants. Les colonnes D/s, □/D et P/D donnent respectivement les moyennes du nombre de nœuds explorés par seconde, du nombre de sous-ensembles inconsistants détectés par nœud et du nombre d'étapes de propagation faites par nœud. Les colonnes L donnent le pourcentage moyen de transformations apprises. Enfin les colonnes Δ_{NB*} et Δ_{SZ*} donnent la différence en pourcentage entre respectivement le nombre et la taille des clauses de compensation générées par AHMAXSAT^{RPO} et AHMAXSAT^{SIR}. Toutes les colonnes ne prennent en compte que les instances résolues par les deux variantes d'AHMAXSAT.

instances classes		AHMAXSAT^{SIR}						AHMAXSAT^{RPO}						Δ_{NB*}	Δ_{SZ*}
		NB	SZ	D/s	□/D	P/D	L	NB	SZ	D/s	□/D	P/D	L		
ms	crafted bipartite	12	3,3	381	73	1132	7%	14,6	3,8	397	71	1104	7%	-18%	-14%
	crafted maxcut	6,7	3,2	3098	18	135	36%	7,7	3,5	3345	18	131	36%	-13%	-9%
	crafted set-covering	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	random highgirth	20,2	4,2	4650	2	39	29%	32,5	5,5	4527	2	36	31%	-38%	-24%
	random max2sat	12,7	3,4	530	45	892	12%	17	4,2	539	42	851	12%	-26%	-19%
	random max3sat	10,8	3,4	1633	20	206	18%	13,5	3,9	1762	19	195	19%	-20%	-12%
	random min2sat	12,8	3,3	565	36	1096	16%	17,5	4,4	532	34	1055	17%	-27%	-25%
pms	crafted frb	7	3,4	3807	5	46	49%	8,6	3,9	3899	5	46	50%	-18%	-12%
	crafted job-shop	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	crafted maxclique	5,6	3,1	2607	6	226	67%	7	3,5	2564	6	225	67%	-20%	-12%
	crafted maxone	10,5	3,1	2815	27	1076	13%	19,7	4,7	2818	24	852	13%	-47%	-34%
	crafted min-enc/kbtree	17,2	3,5	445	61	1005	4%	24,9	4,6	450	55	776	4%	-31%	-23%
	crafted pseudo/miplib	5,8	3,2	-	3	27	31%	6,3	3,4	-	3	29	29%	-9%	-7%
	crafted reversi	47,3	4,6	50	37	3769	17%	114,8	8,3	34	24	2890	24%	-59%	-45%
	crafted scheduling	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	random min2sat	25,7	3,4	118	52	5984	5%	37,5	4,5	110	50	5681	6%	-32%	-25%
	random min3sat	16,9	3,3	383	28	1216	13%	23,8	4,3	388	26	1152	14%	-29%	-22%
	random pmax2sat	12	3,6	438	66	878	20%	17,3	4,6	411	61	815	21%	-30%	-23%
	random pmax3sat/hi	14,9	3,6	1634	16	236	14%	20,1	4,3	1685	15	224	15%	-26%	-16%
wpms	crafted auction	6,2	3,1	1810	74	114	42%	9,2	3,7	1607	84	121	40%	-33%	-15%
	crafted CSG	14,7	3,4	284	0	15	11%	30,2	5,1	236	0	15	12%	-51%	-34%
	crafted frb	6,6	3,4	4456	4	37	53%	7,8	3,8	4444	4	36	54%	-16%	-12%
	crafted min-enc	26,2	4,8	819	71	9969	6%	45,9	7,5	711	47	9577	7%	-43%	-36%
	crafted pseudo/miplib	22,5	3,9	51	100	1506	1%	35,8	6,4	134	76	1087	2%	-37%	-39%
	crafted ramsey	7,1	3,7	25698	4	13	55%	9,3	4,2	26619	4	12	55%	-23%	-12%
	crafted random-net	10,1	3,1	1267	42	127	8%	13,1	3,7	1597	33	94	10%	-23%	-15%
	crafted set-covering	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	crafted wmaxcut	6,7	3,2	610	96	564	34%	7,8	3,6	634	93	558	34%	-15%	-9%
	random wmax2sat	11,8	3,5	102	264	2559	10%	15,9	4,3	94	252	2457	10%	-26%	-19%
	random wmax3sat	11,1	3,4	402	98	591	12%	14,1	3,9	433	90	533	13%	-22%	-13%
	random wpmax2sat	11,7	3,5	298	205	1145	18%	15,3	4,4	231	167	1094	19%	-24%	-21%
	random wpmax3sat/hi	17,3	3,6	445	63	654	6%	24,8	4,4	430	59	596	7%	-30%	-18%
Ensemble		12,2	3,4	1128	73	1420	21%	17,4	4,3	1135	67	1339	21%	-30%	-20%



(a) Temps de résolution



(b) Nombre de nœuds explorés

FIGURE 4.5 – Comparaison détaillée des temps de résolution et du nombre de nœuds explorés par AHMAXSAT^{SIR} et AHMAXSAT^{RPO}. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

et de la taille des clauses de compensation. Enfin, le pourcentage de transformations apprises (colonnes L) est dans l'ensemble équivalent entre les deux variantes du solveur. On peut cependant remarquer de légères variations sur certaines classes d'instances. Ces variations peuvent perturber la lisibilité des autres indicateurs. En effet, l'augmentation du pourcentage d'apprentissage réduit la redondance dans le calcul de la borne inférieure. Par conséquent, le nombre de propagations et de sous-ensembles inconsistants détectés à chaque nœud de l'arbre de recherche diminue. C'est le cas notamment sur les classes d'instances pms/crafted/pseudo/miplib et wpms/crafted/auction.

Pour conclure, ces résultats montrent que l'heuristique SIR réduit efficacement le nombre et la taille des clauses de compensation ajoutées lors de la transformation des sous-ensembles inconsistants. Ces réductions ont plusieurs effets :

1. elles permettent d'augmenter la vitesse d'exploration de l'arbre de recherche,
2. elles permettent sur la plupart des classes d'instances d'améliorer la qualité de la borne inférieure, ce qui réduit le nombre de nœuds de l'arbre de recherche explorés, mais limite voire annule l'effet de (1),
3. elles affectent dans certains cas le pourcentage de transformations apprises par le solveur.

Ces observations soulignent l'interaction étroite entre les différents composants de l'estimation de la borne inférieure : la propagation unitaire, les transformations par max-résolution et l'apprentissage. En outre, elles montrent que l'ordre d'application des étapes de max-résolution a un impact non seulement sur le nombre et la taille des clauses de compensation, mais également sur la capacité de la propagation unitaire à détecter les sous-ensembles inconsistants et même, de manière plus indirecte, sur la quantité d'apprentissage effectué par le solveur.

4.4 Conclusion

Nous avons cherché dans ce chapitre à limiter l'impact d'un des principaux inconvénients de la règle de la max-résolution : l'augmentation de la taille de la formule due aux clauses de compensation. Nous avons montré que l'ordre dans lequel sont appliquées les étapes de max-résolution lors de la transformation des sous-ensembles inconsistants a un impact sur le nombre et la taille des clauses de compensation générées. En particulier, nous avons vu qu'il y a un lien direct entre la taille du résolvant et le nombre et la taille des clauses de compensation générées par chaque opération de max-résolution. Nous avons donc proposé un nouvel ordre d'application de la max-résolution qui réalise en premier les transformations produisant les plus petits résolvantes intermédiaires.

L'étude expérimentale que nous avons réalisée montre que ce nouvel ordre permet de réduire efficacement le nombre et la taille des clauses de compensation ajoutées lors de la transformation des sous-ensembles inconsistants. Elle montre également que la qualité de l'estimation de la borne inférieure s'en trouve améliorée. Cela pousse à s'interroger sur l'impact des transformations sur la capacité de la propagation unitaire à détecter les sous-ensembles inconsistants. On peut également envisager d'autres ordres d'application des étapes de max-résolution, qui prendraient en compte en plus du nombre et de la taille des clauses de compensations d'autres critères pour améliorer la précision de la borne inférieure ou augmenter l'apprentissage réalisé par les solveurs.

Dans le prochain chapitre, nous étudierons les effets du traitement appliqué sur les sous-ensembles inconsistants quand ceux-ci ne répondent pas aux critères d'apprentissage.

Chapitre 5

Max-résolution locale

À chaque nœud de l'arbre de recherche, les solveurs de type séparation et évaluation calculent la borne inférieure en estimant le nombre de sous-ensembles inconsistants disjoints présents dans la formule. Les sous-ensembles inconsistants sont détectés par des méthodes basées sur la propagation unitaire puis transformés pour assurer qu'ils soient disjoints. Les solveurs existants (e.g. WMAXSATZ [LMP07], MINIMAXSAT [HLO08], AKMAXSAT [Küg10]) utilisent deux méthodes de transformation : la suppression temporaire et la transformation dans le sous-arbre par max-résolution.

La première de ces méthodes consiste à simplement supprimer les sous-ensembles inconsistants de la formule pour la durée du calcul de la borne inférieure. La formule résultante de ces transformations n'est pas équivalente à l'originale et peut contenir moins de sous-ensembles inconsistants.

La seconde méthode de transformation est basée sur l'application d'une série d'opérations de max-résolution (cf. section 2.2.3). La formule obtenue est équivalente à l'originale et les transformations sont conservées dans la sous-partie de l'arbre de recherche, agissant ainsi comme une forme limitée d'apprentissage. Cette méthode peut cependant entraîner une augmentation de la taille de la formule et, dans certains cas, une dégradation des performances des solveurs.

Les solveurs existants n'appliquent la transformation basée sur la règle de max-résolution que sur les sous-ensembles inconsistants remplissant certaines conditions : les critères d'apprentissage (cf. section 2.3.4.3). Sur les sous-ensembles inconsistants ne remplissant pas ces critères, c'est-à-dire une large majorité d'entre eux, ils appliquent la suppression temporaire.

Nous nous intéressons dans ce chapitre à la transformation appliquée aux sous-ensembles inconsistants lorsque ceux-ci ne remplissent pas les critères d'apprentissage. La suppression temporaire ne préserve pas l'équivalence de la formule, ce qui peut affecter la qualité de l'estimation de la borne inférieure. En partant de cette hypothèse, nous proposons de remplacer ce traitement par celui basé sur la max-résolution, mais en conservant les changements localement à chaque nœud de l'arbre de recherche (i.e. sans faire d'apprentissage). De cette manière, l'augmentation de la taille de la formule est faible puisque limitée au nœud auquel les transformations sont faites. De plus, les nouvelles transformations n'affectent pas l'efficacité du solveur dans les autres nœuds de l'arbre de recherche puisque les changements sont locaux à chaque nœud de l'arbre. Les bénéfices escomptés sont une amélioration de la qualité de la LB et donc la réduction du nombre de décisions nécessaires pour résoudre les instances, mais aux prix d'un traitement plus coûteux en termes de temps d'exécution et donc d'une réduction de la vitesse d'exploration de l'arbre de recherche.

Le reste de ce chapitre est organisé comme suit. Nous rappelons et détaillons dans la première section la manière dont sont transformés les sous-ensembles inconsistants durant l'estimation de la borne inférieure et nous motivons notre contribution. La seconde section est consacrée à la présentation du nouveau schéma de transformation des sous-ensembles inconsistants. Nous y illustrons son fonctionnement et donnons les détails de son implémentation. La troisième et dernière section est consacrée à l'étude expérimentale de ce nouveau schéma et à l'analyse des résultats obtenus. Les travaux et résultats présentés dans ce chapitre ont été précédemment publiés [AH14c, AH14b]

5.1 Transformation des sous-ensembles inconsistants

Nous rappelons et détaillons dans cette section la manière dont les sous-ensembles inconsistants sont transformés par les solveurs existants lors du calcul de la borne inférieure. Nous présentons tout d'abord l'algorithme général avant de détailler les méthodes de transformation utilisées : la max-résolution dans le sous-arbre et la suppression temporaire.

5.1.1 Algorithme général

Une fois détectés, les sous-ensembles inconsistants doivent être transformés pour s'assurer qu'ils ne soient comptés qu'une fois. À notre connaissance, tous les solveurs existants utilisent le même schéma général de transformation (algorithme 5.1) : ils commencent par diviser le sous-ensemble inconsistant ψ en deux sous-ensembles ψ' et ψ'' selon des critères d'apprentissage donnés (lignes 2-3). Ces critères sont basés sur un ensemble de motifs ou sur la taille des résolvants intermédiaires (cf. section 2.3.4.3). Puis, la partie de ψ remplissant les critères d'apprentissage (ψ') est transformée en appliquant une série d'opérations de max-résolution entre ses clauses (fonction `max_resolution_sous_arbre`, ligne 4) et les changements sont conservés dans la sous-partie de l'arbre de recherche. Les clauses de l'autre partie (ψ'') sont supprimées temporairement de la formule pour la durée de l'estimation de la borne inférieure (fonction `suppression_temporaire`, ligne 5).

Nous détaillons maintenant ces deux méthodes de transformation, en soulignant leurs avantages et inconvénients respectifs.

Algorithme 5.1 : Fonction `transformation_sous_ensemble_inconsistant` par suppression temporaire et max-résolution dans le sous-arbre.

Data : Une formule CNF Φ, un sous-ensemble inconsistant ψ et un graphe d'implications G ayant permis la détection de ψ.
Result : La formule Φ transformée.

```
1 begin
2     ψ' ← sous-ensemble de ψ qui correspond au critère d'apprentissage;
3     ψ'' ← sous-ensemble de ψ qui ne correspond pas au critère d'apprentissage;
4     Φ ← max_resolution_sous_arbre(Φ,ψ',G);
5     Φ ← suppression_temporaire(Φ,ψ'');
```

5.1.2 Max-résolution dans le sous-arbre

La transformation des sous-ensembles inconsistants par max-résolution consiste à appliquer une série d'opérations de max-résolution entre les clauses des sous-ensembles inconsistants. Si m est le poids minimum des clauses d'un sous-ensemble inconsistant ψ, m est soustrait des clauses originales de ψ. Une clause vide (si le sous-ensemble inconsistant est entièrement transformé) et un ensemble de clauses de compensation est ajouté à la formule. Toutes les clauses ajoutées prennent le poids m. Cette méthode est décrite et illustrée dans les sections 2.2.3 et 2.3.4.

La transformation par max-résolution a l'avantage de préserver l'équivalence de la formule. Elle est utilisée par les solveurs existants comme une forme limitée d'apprentissage en conservant les transformations dans la sous-partie de l'arbre de recherche. Elle permet ainsi de rendre le calcul de la borne inférieure plus incrémental et de limiter les redondances dans la détection et le traitement des sous-ensembles inconsistants. Elle a également des inconvénients. Son application est plus coûteuse en terme de temps de calcul que la suppression temporaire. De plus, la taille de la formule transformée peut croître rapidement. Il a été observé expérimentalement [HLO08] que dans certains cas cette méthode de transformation affectait négativement les performances des solveurs dans la sous-partie de l'arbre de recherche. Pour ces raisons, les solveurs existants limitent l'application de ce traitement à quelques cas particuliers par le biais des règles d'apprentissage.

5.1.3 Suppression temporaire

La suppression temporaire consiste à supprimer les sous-ensembles inconsistants de la formule pour éviter qu'ils ne soient comptés plus d'une fois. Pour un sous-ensemble inconsistant ψ de poids m, une clause résolvante vide (si le sous-ensemble inconsistant est entièrement transformé) de poids m est ajoutée à la formule et m est soustrait du poids des clauses originales de ψ (les clauses de poids nul sont supprimées). Cette méthode est décrite et illustrée dans la section 2.3.4.

La suppression temporaire a l'avantage d'être simple à implémenter. Son coût en termes de temps de calcul est faible en comparaison de celui de la max-résolution dans le sous-arbre. Elle n'entraîne pas d'augmentation de la taille de la formule et comme les changements qu'elle apporte ne sont conservés que pour la durée du calcul de la borne inférieure, elle n'affecte pas l'efficacité des solveurs dans les autres nœuds de l'arbre de

recherche.

L'inconvénient majeur de cette méthode est qu'elle ne préserve pas l'équivalence de la formule. En particulier, la formule transformée peut contenir moins de sous-ensembles inconsistants que l'originale, comme le prouve l'exemple suivant.

Exemple 17. Considérons la formule CNF non valuée $\Phi_1 = \{c_1, c_2, c_3, c_4, c_5\}$ avec $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x_1}, \overline{x_2}\}$, $c_4 = \{\overline{x_1}\}$ et $c_5 = \{\overline{x_2}\}$. Φ_1 contient deux sous-ensembles inconsistants disjoints (par exemple $\{c_1, c_4\}$ et $\{c_2, c_5\}$).

Si on applique SUP* sur Φ_1, on obtient la séquence de propagations $\langle x_1@c_1, \overline{x_2}@c_3\rangle$ et la clause c_2 est vide. Le sous-ensemble $\psi_1 = \{c_1, c_2, c_3\}$ est donc inconsistant. L'application de la suppression temporaire sur ψ_1 produit la formule $\Phi'_1 = \{\square, c_4, c_5\}$. On peut remarquer que pour l'interprétation $I = \{\overline{x_1}, \overline{x_2}\}$ on a $\Phi_1|_I = 2 \neq \Phi'_1|_I = 1$ donc la suppression temporaire ne préserve pas l'équivalence de la formule. De plus, un seul sous-ensemble inconsistant a été détecté alors que la formule originale Φ_1 en contient deux.

5.2 La max-résolution locale

Nous présentons dans cette section une nouvelle méthode de transformation des sous-ensembles inconsistant : la max-résolution locale. Nous motivons tout d'abord la création d'une nouvelle méthode de transformation avant d'en détailler le fonctionnement. Nous discutons de ses avantages et inconvénients. Puis nous illustrons son fonctionnement, en le comparant aux méthodes de transformation existantes. Enfin, nous détaillons son implémentation telle qu'elle est réalisée dans notre solveur AHMAXSAT.

5.2.1 Motivations et définition

Nous avons mesuré, sur le benchmark décrit dans la section 2.6, qu'en moyenne 80% des sous-ensembles inconsistants détectés par notre solveur AHMAXSAT ne correspondent pas aux critères d'apprentissage, ni partiellement, ni totalement, et sont donc transformés par suppression temporaire. De plus comme nous l'avons vu dans la section précédente, cette méthode de transformation ne préserve pas l'équivalence de la formule. On peut donc supposer qu'à chaque nœud de l'arbre de recherche des sous-ensembles inconsistants ne sont pas détectés à cause de l'application de la suppression temporaire, affectant ainsi la qualité de l'estimation de la borne inférieure.

Améliorer la qualité de la borne inférieure pourrait donc être bénéfique en termes de performance, même au prix d'un traitement plus coûteux en temps de calcul, si la réduction du nombre de nœuds explorés est supérieure au surcoût en temps d'exécution.

Nous proposons donc de remplacer la suppression temporaire par le traitement basé sur la règle de la max-résolution, mais en ne conservant les changements que pour la durée de l'estimation de la borne inférieure (autrement dit, sans faire d'apprentissage). Nous appelons cette nouvelle méthode *max-résolution locale* (MRL) pour la différencier de la max-résolution dans le sous-arbre. La max-résolution locale combine plusieurs des avantages des autres méthodes de transformation. Elle préserve l'équivalence de la formule, donc davantage de sous-ensembles inconsistants seront détectés et l'estimation de la borne inférieure sera plus précise. Cela entraînera une réduction du nombre de nœuds

explorés de l'arbre de recherche et l'augmentation de la taille de la formule restera limitée puisque les clauses de compensation ne sont ajoutées que temporairement. Enfin, elle n'affectera pas les performances du solveur dans les autres nœuds de l'arbre de recherche puisque les changements qu'elle produit sont locaux au nœud courant. Son seul inconvénient est d'être plus coûteuse en termes de temps de calcul que la suppression temporaire, et donc de réduire la vitesse d'exploration de l'arbre de recherche. Pour que cette méthode soit efficace, il faut que la réduction du nombre de nœuds explorés soit supérieure au surcoût en temps de calcul qu'elle provoque à chaque nœud.

5.2.2 Illustration

Supposons que nous soyons au niveau k de l'arbre de recherche et que nous ayons la formule non-valuée $\Phi = \{c_1, \ldots, c_{14}\}$ avec $c_1 = \{x_1\}$, $c_2 = \{\overline{x}_1, x_4\}$, $c_3 = \{\overline{x}_1, \overline{x}_5\}$, $c_4 = \{\overline{x}_4, x_7\}$, $c_5 = \{x_5, \overline{x}_7\}$, $c_6 = \{x_2\}$, $c_7 = \{\overline{x}_2, x_4\}$, $c_8 = \{x_3\}$, $c_9 = \{\overline{x}_3, x_5\}$, $c_{10} = \{\overline{x}_3, x_6\}$, $c_{11} = \{\overline{x}_6, \overline{x}_7\}$, $c_{12} = \{x_2, \overline{x}_8, \overline{x}_3\}$, $c_{13} = \{\overline{x}_2, \overline{x}_8, x_9\}$ et $c_{14} = \{\overline{x}_2, \overline{x}_9\}$.

a) La suppression temporaire : L'application de SUP* sur Φ conduit à la séquence de propagation suivante : $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (x_1 est propagée par la clause unitaire c_1, puis x_4 par c_2, etc.). La clause c_5 est vide. Le graphe d'implications correspondant à cette situation est montré dans la Fig. 5.1a. Le sous-ensemble inconsistant correspondant à ce conflit est $\{c_1, c_2, c_3, c_4, c_5\}$. Si on supprime ce sous-ensemble inconsistant de Φ, on obtient $\Phi' = \{\square, c_6, \ldots, c_{14}\}$ et toutes les propagations sont défaites.

L'application de SUP* sur Φ' conduit à la séquence de propagations $\langle x_2@c_6, x_4@c_7, \overline{x}_9@c_{14}, x_3@c_8, x_5@c_9, x_6@c_{10}, \overline{x}_7@c_{11}, \overline{x}_8@c_{13}\rangle$. Aucune clause vide n'a été détectée et il n'y a plus de clauses unitaires pour continuer la propagation (Fig. 5.1b).

On peut donc aller au niveau de décision suivant $k+1$. Les clauses supprimées sont restaurées, les propagations sont défaites et une nouvelle décision $x_8 = vrai$ est faite. Comme précédemment, les variables $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ sont propagées, rendant la clause c_5 vide (Fig. 5.1a), et on obtient Φ' après la suppression de le sous-ensemble inconsistant correspondant. L'application de SUP sur Φ' conduit aux propagations $\langle x_2@c_6, x_4@c_7, x_9@c_{13}\rangle$. La clause c_{14} est alors vide (Fig. 5.1c). Le sous-ensemble inconsistant correspondant $\{c_6, c_{13}, c_{14}\}$ est supprimé de la formule et on obtient $\Phi'' = \{\square, \square, c_7, \ldots, c_{12}\}$.

On continue l'application de SUP : $\langle x_3@c_8, x_2@c_{12}, x_5@c_9, x_6@c_{10}, x_4@c_7, \overline{x}_7@c_{11}\rangle$. La formule ne contient plus de clause unitaire (Fig. 5.1d), et on peut passer au niveau de décision suivant. La suppression temporaire a détecté une inconsistance au niveau k et deux au niveau $k+1$

b) La max-résolution dans le sous-arbre : Comme dans l'exemple précédent, l'application de SUP sur Φ conduit aux propagations $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (Fig. 5.1a). La clause c_5 est vide et le sous-ensemble inconsistant correspondant $\{c_1, c_2, c_3, c_4, c_5\}$ peut être traité. L'application de la max-résolution supprime les clauses du sous-ensemble inconsistant de la formule et ajoute une clause vide (le résolvant) et les clauses de compensations suivantes : $c_{15} = \{\overline{x}_4, \overline{x}_5, x_7\}, c_{16} = \{x_4, x_5, \overline{x}_7\}, c_{17} = \{\overline{x}_1, x_4, \overline{x}_5\}$ et $c_{18} = \{x_1, \overline{x}_4, x_5\}$. La Figure 5.2 montre les étapes de max-résolution appliquées durant cette transformation. On obtient la formule $\Phi''' = \{\square, c_6, \ldots, c_{18}\}$.

Les propagations sont défaites et le traitement des clauses unitaires restantes conduit aux propagations $\langle x_2@c_6, x_4@c_7, \overline{x}_9@c_{14}, x_3@c_8, x_5@c_9, x_6@c_{10}, x_7@c_{15}\rangle$. La clause c_{11} est vide (Fig. 5.1e). L'application de la max-résolution supprime de la formule les clauses

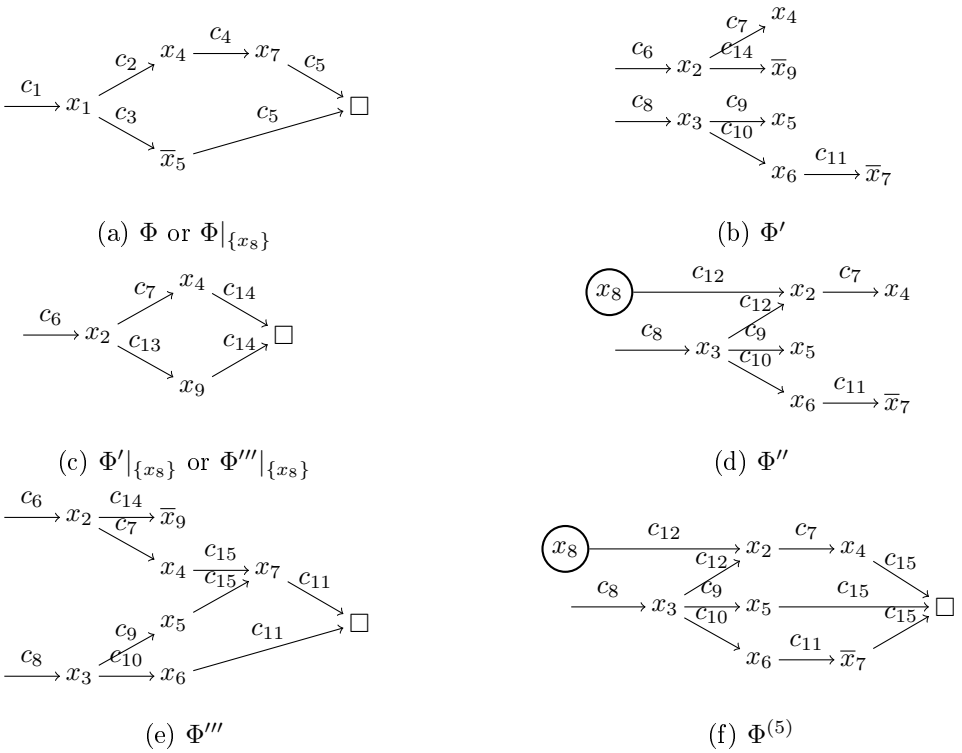


FIGURE 5.1 – Graphe d'implications pour chaque formule de l'exemple. Les nœuds entourés d'un cercle représentent les variables affectées par décisions tandis que les autres nœuds représentent les variables affectées par SUP. Les arcs représentent les clauses utilisées par SUP.

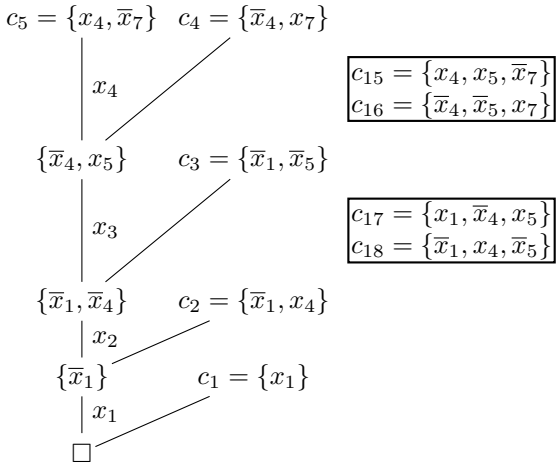


FIGURE 5.2 – Étapes de la max-résolution appliquées sur le sous-ensemble inconsistant $\{c_1, c_2, c_3, c_4, c_5\}$ de la formule Φ. Les clauses de compensations sont encadrées.

du sous-ensemble inconsistant $\{c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{15}\}$ et ajoute une clause vide et les clauses de compensation $c_{19} = \{x_4, \overline{x}_5, \overline{x}_6, \overline{x}_7\}$, $c_{20} = \{x_5, \overline{x}_6, \overline{x}_7\}$, $c_{21} = \{\overline{x}_4, \overline{x}_5, x_6, x_7\}$, $c_{22} = \{x_3, \overline{x}_4, \overline{x}_5, \overline{x}_6\}$, $c_{23} = \{\overline{x}_3, x_4, \overline{x}_5, x_6\}$, $c_{24} = \{\overline{x}_3, x_5, x_6\}$, $c_{25} = \{\overline{x}_3, x_4, x_5\}$, $c_{26} = \{x_3, x_4\}$ et $c_{27} = \{x_2, \overline{x}_4\}$. On obtient $\Phi^{(4)} = \{\square, \square, c_{12}, \ldots, c_{14}, c_{16}, \ldots, c_{27}\}$. Les propagations sont défaites et il ne reste plus de clauses unitaires.

Au niveau de décision $k+1$, les transformations sont conservées et une nouvelle décision $x_8 = \mathit{vrai}$ est réalisée. La formule ne contient pas de clause unitaire, donc le nombre d'inconsistances reste inchangé (deux). La max-résolution dans le sous-arbre a permis de détecter deux inconsistances au niveau k et deux au niveau $k+1$.

c) La max-résolution locale : Comme pour la max-résolution dans le sous-arbre, deux inconsistances sont détectées au niveau k et après transformation on obtient la formule $\Phi^{(4)} = \{\square, \square, c_{12}, \ldots, c_{14}, c_{16}, \ldots, c_{27}\}$. Aucune propagation n'est possible.

Au niveau de décision suivant $k+1$, les propagations sont défaites et la formule originelle est restaurée. Une nouvelle décision $x_8 = \mathit{vrai}$ est réalisée. Les première propagations sont similaires à celles du niveau de décision précédent : $\langle x_1@c_1, x_4@c_2, \overline{x}_5@c_3, x_7@c_5\rangle$ (Fig. 5.1a). Le même conflit est détecté, et comme précédemment sa transformation par max-résolution entraîne la suppression des clauses c_1, c_2, c_3, c_4, c_5 et l'ajout de $\square$, c_{15}, c_{16}, c_{17} et c_{18}. On obtient $\Phi''' = \{\square, c_6, \ldots, c_{18}\}$.

Les propagations sont défaites, et on peut appliquer à nouveau SUP qui conduit aux propagations suivantes : $\langle x_2@c_6, x_4@c_7, x_9@c_{13}\rangle$. La clause c_{14} est vide (Fig. 5.1c). L'application de la max-résolution supprime de la formule les clauses c_6, c_{13}, c_{14} et ajoute une clause vide (aucune clause de compensation n'est ajouté). On obtient la formule $\Phi^{(5)} = \{\square, \square, c_7, \ldots, c_{12}, c_{15}, \ldots, c_{18}\}$.

À nouveau, les propagations sont défaites et on peut traiter les clauses unitaires restantes : $(x_3@c_8, x_5@c_9, x_6@c_{10}, x_2@c_{12}, \overline{x}_7@c_{11}, x_4@c_7)$. La clause c_{15} est vide (Fig. 5.1f), et on peut appliquer la max-résolution sur le sous-ensemble inconsistant correspondant $\{c_7, \ldots, c_{12}, c_{15}\}$. Les clauses originelles sont supprimées, une clause vide $\square$ est ajoutée ainsi que les clauses de compensation $c_{28} = \{x_2, \overline{x}_4, \overline{x}_5, x_7\}$, $c_{29} = \{\overline{x}_2, x_4, x_5, x_7\}$, $c_{30} = \{\overline{x}_2, x_4, \overline{x}_7\}$, $c_{31} = \{\overline{x}_2, \overline{x}_5, x_6, x_7\}$, $c_{32} = \{x_2, \overline{x}_5, \overline{x}_6, \overline{x}_7\}$, $c_{33} = \{x_5, \overline{x}_6, \overline{x}_7\}$, $c_{34} = \{\overline{x}_2, x_3, \overline{x}_5, \overline{x}_6\}$, $c_{35} = \{x_2, \overline{x}_3, x_5, \overline{x}_6\}$, $c_{36} = \{x_2, \overline{x}_3, x_6\}$ et $c_{37} = \{\overline{x}_3, x_5, x_6, \overline{x}_8\}$. On obtient $\Phi^{(6)} = \{\square, \square, \square, c_{16}, \ldots, c_{18}, c_{28}, \ldots, c_{37}\}$. Il n'y a plus de propagations possibles. La max-résolution locale permet de détecter deux inconsistances au niveau k et trois au niveau $k+1$.

Pour conclure cet exemple, la max-résolution locale a permis de détecter une inconsistance de plus que la suppression temporaire aux niveaux k et $k+1$ et une de plus que la max-résolution dans le sous-arbre au niveau $k+1$.

5.2.3 Implémentation

Nous avons implémenté la max-résolution locale dans notre solveur AHMAXSAT. Sans être triviale, son implémentation ne présente pas de difficultés majeures. Tous les sous-ensembles inconsistants sont transformés par max-résolution et seule la portée des changements varie. Sur la partie des sous-ensembles inconsistants vérifiant les critères d'apprentissage (les motifs dans le cas d'AHMAXSAT), les changements sont conservés dans la sous-partie de l'arbre de recherche tandis que les transformations des autres clauses des sous-ensembles inconsistants ne sont conservées que localement au nœud auquel elles ont

été réalisées. L'algorithme 5.2 montre le fonctionnement de cette méthode.

Algorithme 5.2 : Fonction `transformation_sous_ensemble_inconsistant` par max-résolution locale et max-résolution dans le sous-arbre.

Data : Une formule CNF Φ, un sous-ensemble inconsistant ψ et un graphe d'implications G ayant permis la détection de ψ.
Result : La formule Φ transformée.

```
1 begin
2     ψ' ← sous-ensemble de ψ qui correspond au critère d'apprentissage;
3     ψ'' ← sous-ensemble de ψ qui ne correspond pas au critère d'apprentissage;
4     Φ ← max_resolution_sous_arbre(Φ,ψ',G);
5     Φ ← max_resolution_locale(Φ,ψ'',G);
```

Deux listes de changements (locale et dans le sous-arbre) sont maintenues simultanément et les interactions entre ces listes sont gérées minutieusement. Les changements locaux sont défaits à la fin de l'estimation de la borne inférieure : les clauses ajoutées sont supprimées et les changements de poids sont restaurés. La gestion des changements conservés dans la sous-partie de l'arbre de recherche est identique à celle de l'algorithme original : ils sont restaurés lors des retour-arrières dans l'arbre de recherche.

5.3 Étude expérimentale

Nous avons implémenté la max-résolution locale dans notre solveur AHMAXSAT. Nous distinguons deux variantes : AHMAXSAT^{MRL} qui utilise la max-résolution locale et AHMAXSAT^{S} qui utilise le schéma classique de transformation des sous-ensembles inconsistants (qui supprime donc temporairement les sous-ensembles inconsistants ne remplissant pas les critères d'apprentissage). Nous avons comparé ces deux variantes sur le benchmark et selon le protocole présenté en section 2.6.

Les résultats sont présentés dans la table 5.1. La variante utilisant la max-résolution locale résout 49 instances de plus que celle basée sur la suppression temporaire (colonnes S). Sur l'ensemble des instances du benchmark, le nombre moyen de nœuds explorés par AHMAXSAT^{MRL} est réduit de 57,3% par rapport à celui d'AHMAXSAT^{S} (colonnes D) et le temps moyen de résolution est réduit de 49,6% (colonnes T). Ces chiffres masquent cependant de fortes disparités entre les classes d'instances. Ainsi, même si le nombre moyen de nœuds explorés par AHMAXSAT^{MRL} est toujours inférieur ou égal à celui d'AHMAXSAT^{S}, la différence entre les deux varie de +0,8% à -98,2%. De la même manière, la différence en termes de temps moyen de résolution varie de +192,9% a -92,8%.

Sur la majorité des classes d'instances (e.g. ms/crafted/bipartite, ms/random/max2-sat ou pms/crafted/min-enc/kbtree), la réduction du nombre de nœuds explorés est importante et suffit à compenser le surcoût engendré par l'application de la max-résolution locale. Sur d'autres classes d'instances (e.g. ms/random/highgirth, pms/crafted/maxone ou wpms/crafted/auction), le nombre de nœuds explorés est aussi réduit mais dans des proportions moindres. Par conséquent, le temps de résolution n'est pas ou peu amélioré voire dans certains cas augmenté.

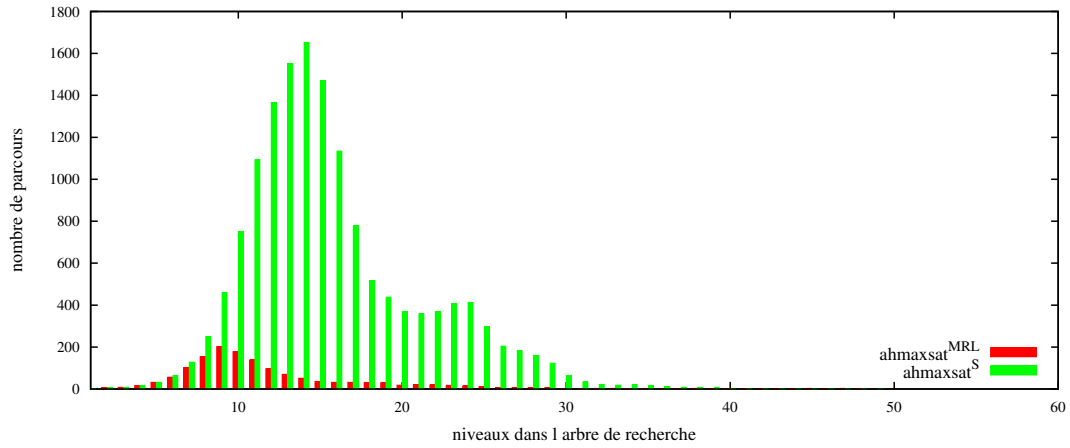
La figure 5.3 montre le nombre de nœuds explorés par niveaux de l'arbre de recherche sur deux instances du benchmark. On peut voir que sur l'instance ms/random/

TABLE 5.1 – Comparaison des variantes AHMAXSAT^{MRL} et AHMAXSAT^{S}. Les deux première colonnes montrent les classes d'instances et le nombre d'instances par classe. Pour chaque variante, les colonnes S, D et T donnent respectivement le nombre d'instances résolues, le nombre moyens de nœuds explorés et le temps moyen de résolution en secondes. Les colonnes marquées avec une étoile ne prennent en compte que les instances résolues par toutes les variantes.

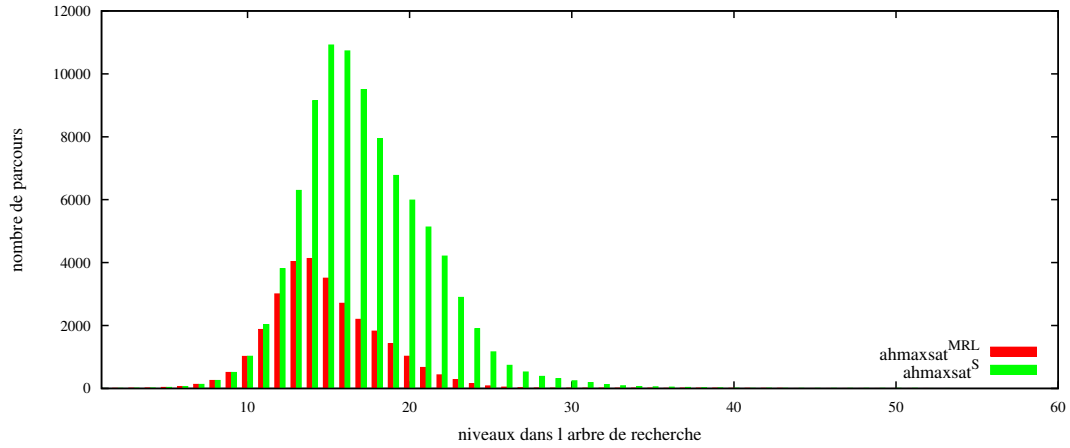
instances classes		#	AHMAXSAT^{MRL}			AHMAXSAT^{S}			Δ_{D*}	Δ_{T*}
			S	D*	T*	S	D*	T*		
ms	crafted bipartite	100	**100**	33331	**86,6**	99	421359	373,2	-92,1%	-76,8%
ms	crafted maxcut	67	**56**	173361	**40,3**	55	254783	43,1	-32,0%	-6,6%
ms	crafted set-covering	10	0	-	-	0	-	-	-	-
ms	random highgirth	82	7	5016310	1166,0	7	7665094	**1140,9**	-34,6%	2,2%
ms	random max2sat	100	100	45444	**89,5**	100	261626	207,5	-82,6%	-56,9%
ms	random max3sat	100	100	340968	**231,2**	100	725170	248,6	-53,0%	-7,0%
ms	random min2sat	96	96	1046	**2,6**	96	14579	13,6	-92,8%	-81,1%
pms	crafted frb	25	5	585949	**154,2**	5	819455	167,6	-28,5%	-8,0%
pms	crafted job-shop	3	0	-	-	0	-	-	-	-
pms	crafted maxclique	158	**133**	19493	**18,7**	132	30162	20,3	-35,4%	-7,7%
pms	crafted maxone	140	109	133560	44,6	**110**	160551	**32,1**	-16,8%	38,8%
pms	crafted min-enc/kbtree	42	**34**	53070	**95,3**	15	1045226	718,1	-94,9%	-86,7%
pms	crafted pseudo/miplib	4	2	288	<0,1	2	1047	<0,1	-72,5%	-15,4%
pms	crafted reversi	44	**8**	4067	**55,0**	7	25424	122,1	-84,0%	-54,9%
pms	crafted scheduling	5	0	-	-	0	-	-	-	-
pms	random min2sat	60	**58**	3146	**37,4**	44	61031	515,6	-94,8%	-92,8%
pms	random min3sat	60	**58**	66357	**179,4**	52	344180	436,3	-80,7%	-58,9%
pms	random pmax2sat	60	60	1021	**3,2**	60	3213	3,9	-68,2%	-17,9%
pms	random pmax3sat/hi	30	30	84944	**60,1**	30	232411	68,9	-63,5%	-12,8%
wpms	crafted auction	40	40	310021	115,5	40	326623	**90,6**	-5,1%	27,5%
wpms	crafted CSG	10	4	54813	452,3	4	54359	**434,7**	0,8%	4,0%
wpms	crafted frb	34	14	212369	**58,7**	14	296987	68,1	-28,5%	-13,7%
wpms	crafted min-enc	74	**64**	20611	18,5	58	13922	**10,0**	48,1%	84,5%
wpms	crafted pseudo/miplib	12	4	1505	**158,8**	4	7062	293,3	-78,7%	-45,9%
wpms	crafted ramsey	15	4	157559	44,8	**5**	186329	**15,3**	-15,4%	192,9%
wpms	crafted random-net	32	3	802991	572,4	3	2126393	**498,6**	-62,2%	14,8%
wpms	crafted set-covering	45	25	3754	**46,4**	25	205372	100,6	-98,2%	-53,9%
wpms	crafted wmaxcut	48	**46**	28064	41,6	44	35438	**38,9**	-20,8%	6,8%
wpms	random wmax2sat	120	120	3993	**45,4**	120	46303	161,2	-91,4%	-71,8%
wpms	random wmax3sat	40	40	36836	99,5	40	82859	**74,3**	-55,5%	34,0%
wpms	random wpmax2sat	90	90	575	**5,5**	90	1307	5,8	-56,0%	-4,1%
wpms	random wpmax3sat/hi	30	30	30451	84,1	30	92583	**63,6**	-67,1%	32,2%
Ensemble		1776	**1440**	99223	**72,3**	1391	232238	143,4	-57,3%	-49,6%

TABLE 5.2 – Statistiques détaillées des variantes AHMAXSAT^{MRL} et AHMAXSAT^{S}. Les colonnes □/D et P/D donnent respectivement les moyennes du nombre de sous-ensembles inconsistants et de propagations faites par nœud de l'arbre de recherche. Les colonnes %L donnent le pourcentage d'apprentissage réalisé. Les deux dernières colonnes $|\Phi|_{avg}$ et $|\Phi|_{max}$ donnent respectivement la moyenne et le maximum du nombre de clauses temporaires ajoutées par la max-résolution locale à chaque nœud de l'arbre de recherche, exprimé en pourcentage de la taille de la formule originale.

instances classes			AHMAXSAT^{MRL}			AHMAXSAT^{S}			$\|\Phi\|_{avg}$	$\|\Phi\|_{max}$
			□/D*	P/D*	%L	□/D*	P/D*	%L		
ms	crafted	bipartite	73,3	1131,5	7%	40,4	778,8	15%	64,8%	133,9%
		maxcut	15,5	99,6	36%	12,5	77,5	43%	10,9%	39,5%
		set-covering	-	-	-	-	-	-	-	-
	random	highgirth	2,5	43,3	27%	2,1	35,5	37%	5,1%	84,7%
		max2sat	44,5	891,7	12%	25,8	598,6	19%	37,1%	94,5%
		max3sat	20,1	206,1	18%	15,9	161,4	25%	21%	64,2%
		min2sat	35,7	1096,1	16%	19,3	711,3	25%	22,2%	63%
pms	crafted	frb	4,8	46,1	49%	3,9	35,5	58%	0,4%	4,3%
		job-shop	-	-	-	-	-	-	-	-
		maxclique	5,9	226,5	67%	4,9	210,4	73%	2%	9,7%
		maxone	23,3	714	13%	14,5	369,5	21%	30%	210,9%
		min-enc/kbtree	57,8	967,2	4%	36,3	560,7	11%	53,5%	170,4%
		pseudo/miplib	3,2	27,1	31%	1,8	11,5	54%	11,3%	89,2%
		reversi	33,2	4287,6	14%	7,6	2010,2	39%	25,2%	98,2%
		scheduling	-	-	-	-	-	-	-	-
	random	min2sat	47,2	5510,6	5%	26,3	7057,8	17%	24,6%	104,5%
		min3sat	27,3	1203,6	13%	15,5	835,5	25%	10%	29,7%
		pmax2sat	65,8	878,4	20%	46,4	568,8	29%	19,7%	218,1%
		pmax3sat/hi	15,6	235,8	14%	10,8	166,6	24%	29,1%	99,6%
wpms	crafted	auction	73,6	114,4	42%	41,7	65,8	51%	20,4%	250,9%
		CSG	<0,1	14,8	11%	<0,1	17,1	17%	<0,1%	4%
		frb	4,3	36,8	53%	3,4	27,9	63%	1,1%	7,8%
		min-enc	74,3	10134	6%	13,6	7719,7	12%	30,5%	161,5%
		pseudo/miplib	94,5	35734,3	1%	8,9	28135,8	14%	103,5%	1101,7%
		ramsey	4,2	13,1	55%	2,7	10,4	65%	6,9%	689,5%
		random-net	40,7	123,9	8%	27,7	83,9	11%	30,2%	887,8%
		set-covering	48,9	476,6	4%	42,8	191,3	13%	200,4%	833,3%
		wmaxcut	66	205,6	35%	58,5	183,7	41%	29,3%	167,8%
	random	wmax2sat	262,9	2552,9	10%	159,4	1690,3	15%	206,1%	397,2%
		wmax3sat	97,7	591,3	12%	71	421,5	17%	110,7%	255,9%
		wpmax2sat	205,5	1144,6	18%	141,9	898,8	28%	57,2%	400%
		wpmax3sat/hi	63,4	654	6%	40,7	419,4	13%	135,9%	365,3%
Ensemble			71,1	1417,1	21%	44,2	1095,3	29%	49,7%	172,8%



(a) Instance ms/random/max2sat/s2v100c1200-10.cnf



(b) Instance wpms/random/wmax3sat/s3v90c700-5.wcnf

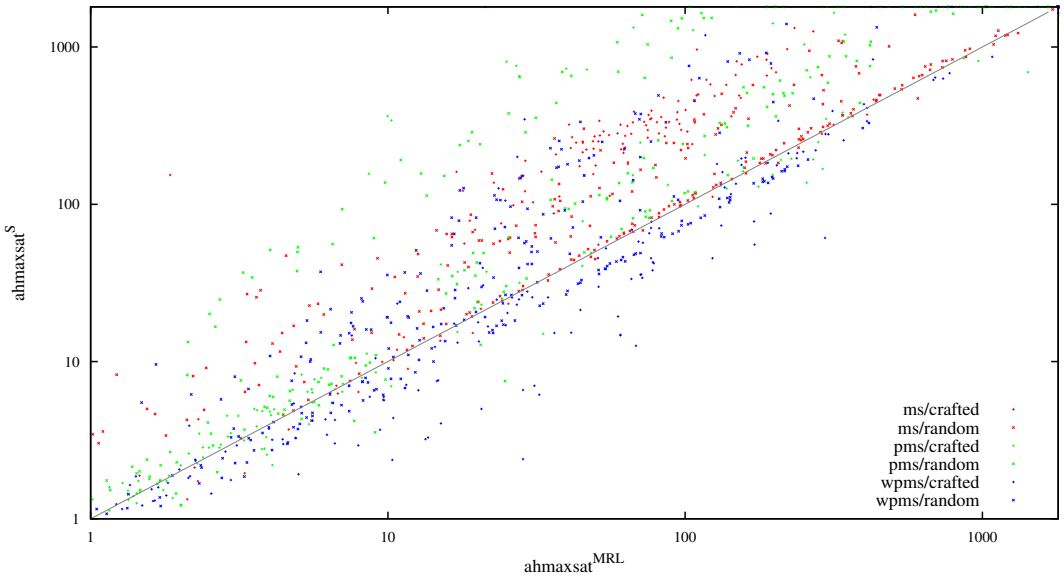
FIGURE 5.3 – Comparaison du nombre de nœuds explorés par AHMAXSAT^{MRL} et AHMAXSAT^{S} à chaque niveau de l'arbre de recherche.

max2sat/s2v100c1200-10.cnf (figure 5.3a) le "pic", c'est-à-dire plus grand nombre de nœuds explorés, se situe au niveau 9 de l'arbre de recherche pour AHMAXSAT^{MRL} et au niveau 14 pour AHMAXSAT^{S}. Cela montre que la max-résolution locale permet à notre solveur de faire des coupures plus haut dans l'arbre de recherche, et donc de réduire sensiblement le nombre de nœuds explorés. Sur l'instance wpms/random/wmax3sat/s3v90c700-5.wcnf (figure 5.3b), les pics d'AHMAXSAT^{MRL} et d'AHMAXSAT^{S} se situent respectivement aux niveaux 15 et 17 de l'arbre de recherche. L'écart entre les pics étant moins important, la réduction du nombre de nœuds explorés est plus faible.

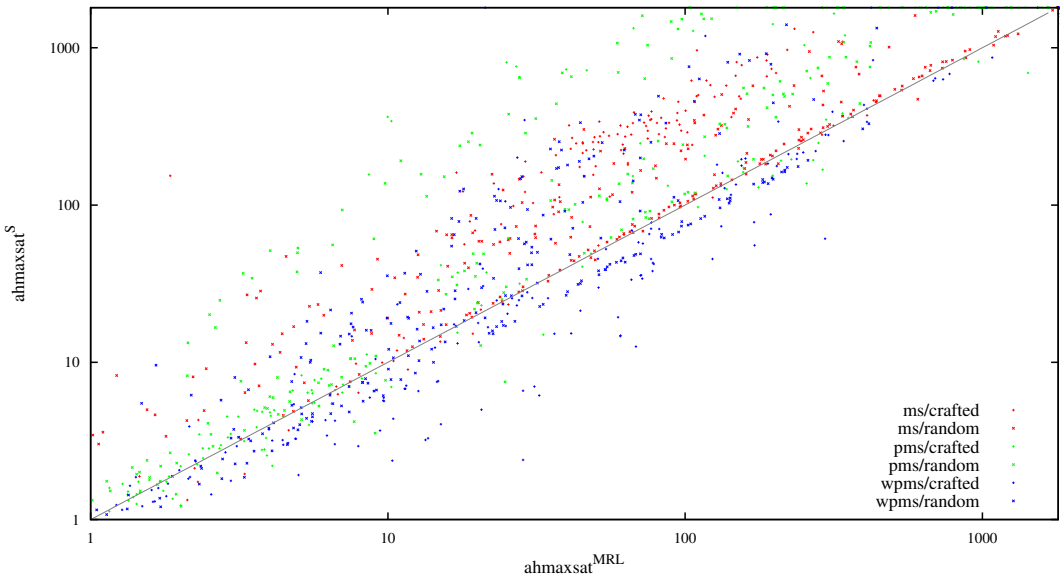
La table 5.2 présente les nombres moyens de propagations (colonnes P/D) et de sous-ensembles inconsistants détectés (colonnes □/D) à chaque nœud de l'arbre de recherche ainsi que le pourcentage moyen d'apprentissage réalisé (colonnes %L). Sur l'ensemble du benchmark, la max-résolution locale permet de réaliser à chaque nœud de l'arbre de recherche davantage de propagations unitaires (+60,8%) et donc de détecter davantage de sous-ensembles inconsistants (+29,3%). On peut cependant remarquer que sur certaines des classes d'instances où la max-résolution locale n'améliore pas les performances du solveur, comme wpms/crafted/auction, l'augmentation de ces valeurs est importante (respectivement +73,8% et +76,5%) sans que cela ne se reflète sur la réduction du nombre de nœuds explorés (-5,1%). Par conséquent, l'exploration de l'arbre de recherche est plus lente et le temps de résolution augmente (+21,6%).

Ce phénomène est une conséquence de la réduction de l'apprentissage (en moyenne -26,7%) entraînée par la max-résolution locale. Deux facteurs peuvent expliquer cette réduction. Tout d'abord, l'apprentissage est probablement réalisé majoritairement dans les nœuds bas de l'arbre de recherche, lorsque de nombreuses clauses ont été réduites par les affectations. Comme la max-résolution locale réduit le nombre de nœuds explorés, le solveur passe moins de temps dans les branches basses de l'arbre de recherche et réalise donc moins d'apprentissage. Ensuite, les clauses de compensation ajoutées par la max-résolution locale ne sont conservées que pour la durée de l'estimation de la borne inférieure. Par conséquent, il suffit qu'un sous-ensemble inconsistant contienne une de ces clauses pour qu'on ne puisse pas apprendre sa transformation.

Il est intéressant d'observer que le nombre de clauses ajoutées par l'application de la max-résolution locale est limité (table 5.2, colonnes $|\Phi|_{avg}$ et $|\Phi|_{max}$). Sur l'ensemble du benchmark, le nombre moyen de clauses temporaires ajoutées à chaque nœud de l'arbre représente 49,7% de la taille de la formule originale. Le nombre maximal de clauses ajoutées à un nœud est lui aussi raisonnable (172,8% du nombre de clauses de la formule originale). Là encore, ces valeurs varient selon les classes d'instances. Sur les instances non valuées (ms et pms), le nombre moyen de clauses ajoutées dépasse rarement 50% du nombre de clauses originales tandis que le nombre maximal de clauses ajoutées est toujours inférieur à 250%. Sur les instances valuées (wpms), ces valeurs sont beaucoup plus élevées. Le nombre moyen de clauses atteint 200% de l'original tandis que les maximums dépassent parfois les 1000%. Ces différences entre instances valuées et non valuées peuvent s'expliquer par un phénomène d'"étalement" des clauses. En effet, une clause de poids m peut-être utilisée dans m sous-ensembles inconsistants (s'ils sont tous de poids 1). Dans cette situation, des clauses de compensation de poids 1 seront générées lors de la transformation des m sous-ensembles inconsistants, provoquant une fragmentation ou une division des informations contenues dans la clause originale dans de nombreuses nouvelles clauses.



(a) Temps de résolution



(b) Nombre de nœuds explorés

FIGURE 5.4 – Comparaison détaillée des temps de résolution et du nombre de nœuds explorés par AHMAXSAT^{MRL} et AHMAXSAT^{S}. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

Les figures 5.4a et 5.4b comparent instance par instance le nombre de nœuds explorés et le temps de résolution des deux variantes de notre solveur. On peut voir que le nombre de nœuds est réduit sur quasiment toutes les instances du benchmark, mais que sur certaines classes d'instances cette réduction ne suffit pas à compenser le surcoût en temps de traitement, par conséquent le temps de résolution augmente.

5.4 Conclusion

Nous avons présenté dans ce chapitre une nouvelle méthode de traitement des sous-ensembles inconsistants qui permet une meilleure estimation de la borne inférieure et donc la réduction du nombre de nœuds examinés dans l'arbre de recherche, mais au prix d'un traitement plus coûteux en termes de temps d'exécution. Les résultats de l'étude expérimentale que nous avons conduite montrent que cette nouvelle méthode permet de réduire le nombre de nœuds explorés pour quasiment toutes les instances considérées et le temps de résolution sur une large majorité d'entre elles.

Nous avons vu également que la max-résolution locale a des inconvénients : elle est plus coûteuse en temps de calcul et elle réduit l'apprentissage réalisé apr le solveur. Une perspective d'amélioration consisterait à ne pas appliquer systématiquement la max-résolution locale, en déterminant dans quels cas elle est utile et dans quels cas elle affecte négativement le performances du sovleur.

Dans le prochain chapitre, nous proposons de nouveaux motifs permettant d'étendre les critères d'apprentissages utilisés par les solveurs de type séparation et évaluation.

Chapitre 6

Augmentation de l'apprentissage

Le calcul de la borne inférieure, et en particulier l'estimation de la somme des poids des sous-ensembles inconsistants disjoints présents dans la formule, est un des composants primordiaux des solveurs de type séparation et évaluation. Il est effectué à chaque nœud de l'arbre de recherche et représente une part importante du temps de calcul total des solveurs. La qualité de cette estimation détermine le nombre de nœuds de l'arbre de recherche qui sont explorés. De plus, nous avons vu que ce calcul est très redondant : les mêmes sous-ensembles inconsistants peuvent être détectés à plusieurs nœuds de l'arbre de recherche. L'apprentissage semble donc être un moyen naturel de réduire le temps de calcul consacré à l'estimation de la borne inférieure en limitant la redondance sans pour autant affecter la qualité de l'estimation.

Les solveurs existants mémorisent (complètement ou partiellement) les transformations par max-résolution des sous-ensembles inconsistants lorsque ceux-ci correspondent (complètement ou partiellement) à certains motifs prédéfinis (cf. section 2.3.4). Nous donnons dans ce chapitre une définition générale des motifs dont la transformation produit des clauses résolvantes unitaires (*unit clause subset*, UCS). La mémorisation de telles transformations permet d'améliorer la capacité de la propagation unitaire à détecter des sous-ensembles inconsistants. La qualité de la borne inférieure s'en trouve donc améliorée tout en rendant son calcul plus incrémental. On peut noter que des règles d'inférences correspondant à ce type de motifs ont déjà été étudiées par le passé [BR99, LH05a, LHdG08, LMMP09]. Nous étudions expérimentalement l'impact de l'ajout de ces nouveaux motifs aux critères d'apprentissage existants en faisant varier leur taille

et la taille des clauses qui les composent. Les résultats obtenus montrent l'intérêt de ces motifs ainsi que leurs limites.

Ce chapitre est organisé comme suit. Nous détaillons dans la première partie les schémas d'apprentissage existants. La seconde partie est consacrée à la présentation des nouveaux ensembles de motifs. Nous discutons de leur détection et étudions leurs fréquences d'apparition dans les instances du benchmark présenté en section 2.6. Dans la dernière partie de ce chapitre, nous étudions expérimentalement l'impact de l'utilisation de ces motifs dans le schéma d'apprentissage d'AHMAXSAT. Les travaux et résultats présentés dans ce chapitre ont été précédemment publiés [AH14e, AH15d]

6.1 Calcul de la borne inférieure et apprentissage

6.1.1 Détection et transformation des sous-ensembles inconsistants

À chaque nœud de l'arbre de recherche, les solveurs de type séparation et évaluation calculent la borne inférieure en estimant la somme des poids des clauses qui seront falsifiées dans la branche courante de l'arbre de recherche. Pour ce faire, ils additionnent les poids des sous-ensembles inconsistants disjoints présents dans la formule au nœud courant de l'arbre de recherche. Les sous-ensembles inconsistants sont détectés par des méthodes basées sur la propagation unitaire (propagation unitaire simulée, SUP, et méthode des littéraux contradictoires, FL, cf. section 2.3.4) puis transformés pour s'assurer qu'ils ne sont comptés qu'une fois. Une des méthodes de transformation existantes est basée sur la règle de la max-résolution (cf. section 2.2.3). Pour un sous-ensemble inconsistant ψ de poids m (c'est-à-dire dont le minimum des poids des clauses est m), elle consiste à soustraire m du poids des clauses de ψ (les clauses de poids nul sont supprimées) et à ajouter à la formule une clause vide de poids m et un ensemble de clauses de compensation. La formule ainsi obtenue est équivalente à l'originale.

Après transformation par max-résolution, les sous-ensembles inconsistants sont présents dans la formule sous forme de clauses vides. Ils peuvent être directement pris en compte dans le calcul de la borne inférieure, sans nécessiter ni détection par propagation unitaire ni transformation. Par conséquent, les solveurs peuvent réaliser une forme d'apprentissage en conservant ces transformations d'un nœud à l'autre, rendant ainsi le calcul de la borne inférieure plus incrémental. Cependant, comme nous l'avons vu en section 2.3.4.3 et dans le chapitre précédent, les transformations par max-résolution augmentent la taille de la formule. De plus, il a été observé expérimentalement [HLO08] que les performances des solveurs peuvent être affectées négativement par ces transformations. Pour ces raisons, les transformations par max-résolution ne sont conservées que dans la sous-partie de l'arbre de recherche et uniquement lorsque les sous-ensembles inconsistants transformés remplissent des critères prédéfinis. Nous rappelons ces critères dans la suite de cette section.

6.1.2 Schémas d'apprentissage

Deux types de schéma d'apprentissage existent, qui définissent dans quels cas les transformations par max-résolution doivent être mémorisées. Le premier, utilisé dans le solveur MINIMAXSAT [HLO08], est basé sur la taille des clauses résolvantes intermédiaires. Les transformations des sous-ensembles inconsistants ne sont conservés dans la

sous-partie de l'arbre de recherche que si tous les résolvants intermédiaires obtenus durant leurs transformations sont de taille inférieure à quatre.

Le second schéma d'apprentissage, utilisé dans les solveur WMAXSATZ [LMP07], AKMAXSAT [Küg10] et dans notre solveur AHMAXSAT, est basé sur un ensemble de motifs. Ne sont conservées que les transformations des parties des sous-ensembles inconsistants qui correspondent à ces motifs. Les principaux motifs utilisés sont les suivants (avec en haut les clauses originales des sous-ensembles inconsistants et en bas les clauses obtenues après transformation) :

$$\frac{\{l_1, l_2\}, \{l_1, \overline{l_2}\}}{\{l_1\}} \tag{2.15}$$

$$\frac{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}} \tag{2.16}$$

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-1}}, l_k\}, \{\overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-1}, \overline{l_k}\}} \tag{2.17}$$

Ces motifs peuvent être combinés. Ainsi, on obtient par 2.15 et 2.17 :

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-2}}, l_{k-1}\}, \{l_{k-1}, l_k\}, \{l_{k-1}, \overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-2}, \overline{l_{k-1}}\}} \tag{2.18}$$

et par 2.16 et 2.17 :

$$\frac{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-3}}, l_{k-2}\}, \{l_{k-2}, l_{k-1}\}, \{l_{k-2}, l_k\}, \{\overline{l_{k-1}}, \overline{l_k}\}}{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-3}, \overline{l_{k-2}}\}, \{l_{k-2}, l_{k-1}, l_k\}, \{\overline{l_{k-2}}, \overline{l_{k-1}}, \overline{l_k}\}} \tag{2.19}$$

Ces motifs ne couvrent pas nécessairement toutes les clauses des sous-ensembles inconsistants. Seule la partie de la transformation concernant les clauses couvertes par les motifs est apprise. On peut remarquer que les transformations des sous-ensembles inconsistants correspondant aux motifs 2.15 et 2.16 produisent des clauses résolvantes unitaires.

6.2 Étendre l'apprentissage

Nous donnons dans cette section une définition générale des motifs produisant après transformation des clauses résolvantes unitaires (*unit clause subset*, UCS). Nous motivons cette généralisation et montrons que ces motifs peuvent être détectés efficacement. Nous discutons également de la fréquence d'apparition de ces motifs dans les sous-ensembles inconsistants des instances du benchmark présenté en section 2.6.

6.2.1 UCS

Nous avons vu dans la section précédente que deux des motifs utilisés dans les schémas d'apprentissage des solveurs existants produisent après transformation des clauses résolvantes unitaires. L'intérêt de la mémorisation de ces transformations est double. D'un coté, elle permet de limiter la redondance des étapes de propagation unitaire réalisées dans la sous-partie de l'arbre de recherche et, d'un autre côté, les clauses unitaires

produites peuvent permettre de détecter plus de sous-ensembles inconsistants par propagation unitaire, et donc améliorer la qualité de la borne inférieure.

Nous proposons dans ce chapitre d'étendre l'apprentissage réalisé par les solveurs en prenant en compte de nouveaux motifs produisant après transformation des clauses résolvantes unitaires. Ces motifs peuvent être définis formellement comme suit.

Définition 37 (Sous-ensemble produisant des clauses résolvantes unitaires (*unit clause subsets*, UCS))**.** Considérons une formule CNF Φ. Un sous-ensemble produisant des clauses résolvantes unitaires est un ensemble $\{c_1, \ldots, c_k\} \subset \Phi$ avec $\forall i \in \{1, \ldots, k\}$, $|c_i| > 1$ et tel qu'il existe une séquence d'étapes de max-résolution sur les clauses $c_1, \ldots, c_k$ qui produit une clause résolvante unitaire. Nous notons k-UCS l'ensemble des motifs correspondants aux UCS de taille k.

Exemple 18. Ci-dessous les motifs de l'ensemble 3-UCS :

$$\frac{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}} \tag{6.1}$$

$$\frac{\{l_1, l_2\}, \{\overline{l_2}, l_3\}, \{l_1, \overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{\overline{l_1}, \overline{l_2}, l_3\}} \tag{6.2}$$

$$\frac{\{l_1, l_2\}, \{l_1, l_3\}, \{l_1, \overline{l_2}, \overline{l_3}\}}{\{l_1\}, \{l_1, l_2, l_3\}} \tag{6.3}$$

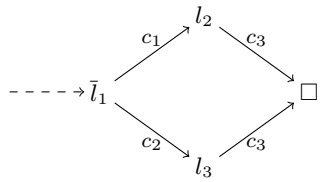
$$\frac{\{l_1, l_2\}, \{l_1, \overline{l_2}, l_3\}, \{l_1, \overline{l_2}, \overline{l_3}\}}{\{l_1\}} \tag{6.4}$$

Les graphes d'implications décrivant des exemples d'étapes de propagation pouvant mener à la détection de ces motifs sont présentés dans la figure 6.1.

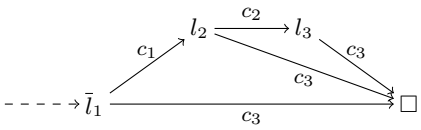
Puisque l'un des objectifs de l'apprentissage des transformations des UCS est d'accroître le nombre d'affectations réalisées par propagation unitaire (SUP ou FL) lors du calcul de la borne inférieure, nous ne considérons pas les sous-ensembles contenant des clauses unitaires. Dans le meilleur des cas (s'ils contiennent uniquement une clause unitaire), la transformation de tels sous-ensembles laisserait inchangé le nombre de clauses unitaires présentes dans la formule. Dans le pire des cas (s'ils contiennent plus d'une clause unitaire), la formule transformée contiendrait moins de clauses unitaires que l'originale. Le nombre d'affectations faites par propagation unitaire et par conséquent le nombre de sous-ensembles inconsistants détectés pourrait alors être réduit.

Nous faisons une distinction entre les motifs des ensembles k-UCS selon la taille de leurs clauses. Nous notons k^b-UCS le sous-ensemble de k-UCS composé des motifs qui ne contiennent que des clauses binaires et k^t-UCS le sous-ensemble composé des motifs contenant au moins une clause ternaire. Pour tout $j > 0$, nous noterons $\{k_1, k_2, \ldots, k_j\}$-UCS l'union des ensembles k_1-UCS, k_2-UCS, ..., k_j-UCS.

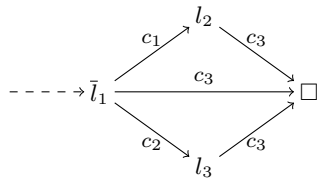
On peut noter que les motifs 2.15 et 2.16 inclus dans les schémas d'apprentissage existants (cf. section 2.3.4.3 ou 6.1.2) appartiennent respectivement aux ensembles 2-UCS et 3^b-UCS.



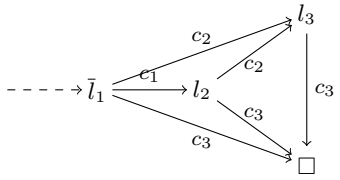
(a) motif 6.1



(b) motif 6.2



(c) motif 6.3



(d) motif 6.4

FIGURE 6.1 – Graphes d'implications décrivant des exemples de séquences de propagations unitaires permettant de détecter les motifs de l'ensemble des 3-UCS présentés dans l'exemple 18. On suppose que le littéral $\overline{l_1}$ a déjà été propagé et que les clauses de chacun des motifs sont nommées c_1, c_2 et c_3.

6.2.2 Détection des k-UCS

Les motifs des k-UCS sont facilement détectables lors de l'analyse du graphe d'implications réalisé pour construire les sous-ensembles inconsistants. En effet, les clauses situées entre le conflit et le premier point d'implication unique [1] (*first unit implication point*, FUIP) [MSS99] produisent, lorsqu'elles sont transformées par max-résolution, un résolvant unitaire. Cette analyse peut être réalisée comme suit. Les littéraux falsifiés de la clause conflictuelle sont ajoutés à une queue Q. À chaque étape, le littéral de Q propagé le plus récemment est supprimé de Q et les littéraux falsifiés de sa source de propagation sont ajoutés à Q. Le FUIP est trouvé lorsque Q contient un seul littéral. Il suffit de compter le nombre et les tailles des clauses rencontrées (la clause conflictuelle et les sources de propagation) avant le FUIP pour savoir s'il on est en présence d'un UCS valide. Cela ne change pas la complexité de la procédure d'analyse des conflits (cf. propriété 3) et le surcoût en termes de temps de calcul est négligeable.

6.2.3 Fréquences d'apparition des k-UCS

Nous avons mesuré le taux d'apparition des motifs k-UCS dans les sous-ensembles inconsistants détectés par AHMAXSAT sur les instances du benchmark présenté en section 2.6. Les résultats sont présentés dans la table 6.1. Il est intéressant d'observer qu'en moyenne, les motifs utilisés dans le schéma le plus répandu d'apprentissage (2-UCS et 3^b-UCS) sont présents dans moins de 5% des sous-ensembles inconsistants. En moyenne, des UCS sont détectés dans plus 57% des sous-ensembles inconsistants détectés par AHMAXSAT, ce qui montre que la prise en compte des UCS dans le schéma d'apprentissage

1. C'est-à-dire le littéral propagé le plus récemment et tel que tous les chemins allant des sources du graphe d'implications vers la clause vide passent par lui.

peut potentiellement réduire sensiblement les redondances dans le calcul de la borne inférieure.

On peut remarquer que ces valeurs varient fortement selon les classes d'instances. Par exemple, le taux moyen d'apparition des UCS est inférieur à 3,5% sur les instances de la classe wpms/crafted/set-covering tandis qu'il est supérieur à 85% sur celles de la classe ms/crafted/bipartite. On peut aussi noter que les taux d'apparition de chaque sous-ensemble des UCS varient également d'une classe d'instances à une autre. Cela met en exergue les différences structurelles entre les instances de ces classes.

TABLE 6.1 – Pourcentages d'apparition des motifs k-UCS dans les sous-ensembles inconsistants détectés par AHMAXSAT.

Classes d'instances			2	3^b	3^t	4^b	4^t	5^b	5^t	> 5
ms	crafted	bipartite	0,05	0,52	0,03	0,22	0,08	42,19	0,29	42,13
		maxcut	0,06	15,71	0,04	4,15	9,45	5,24	6,73	21,49
		set-covering	-	-	-	-	-	-	-	-
	random	highgirth	1,47	0,79	0,49	0,58	0,8	0,34	1,15	23,54
		max2sat	0,12	3	0,05	20,03	0,59	10,63	2,57	41,98
		max3sat	1,23	3,11	1,45	4,5	4,3	1,92	6,79	38,02
		min2sat	0,39	2,76	0,11	16,93	0,29	12,18	0,9	43,2
pms	crafted	frb	0,11	12,63	0,03	0,02	25,38	2,52	3,05	19,16
		job-shop	-	-	-	-	-	-	-	-
		maxclique	0,09	10,25	0,05	0,03	9,33	3,76	0,95	17,33
		maxone	1,89	0,32	0,56	0,25	0,82	0,15	0,47	5,3
		min-enc/kbtree	0,21	0,74	0,15	2,74	0,45	1,34	1,48	20,41
		pseudo/miplib	0,42	1,17	0,26	0,11	0,67	0,18	5	9,48
		reversi	0,51	2,62	0,37	0,43	2,3	1,45	1,34	29,39
		scheduling	-	-	-	-	-	-	-	-
	random	min2sat	0,57	0,23	0,33	0,12	0,28	6,66	0,28	60,09
		min3sat	0,18	2,33	0,08	0,15	0,66	11,69	0,33	51,22
		pmax2sat	0,96	8,03	0,14	15,92	1,04	5,88	3,43	29,53
		pmax3sat/hi	1,11	1,9	0,65	3,12	1,78	1,85	3,17	37,25
wpms	crafted	auction	0,35	8,73	0,24	0,03	15,82	0,16	3,26	3,79
		CSG	0,16	3,66	0,01	0,05	0,41	>0,01	0,16	1,53
		frb	0,11	13,46	0,04	0,03	21,05	3,04	2,69	19,37
		min-enc	0,29	0,94	0,47	1,4	0,66	0,3	1,17	16,45
		pseudo/miplib	0,27	0,5	0,8	0,28	2,6	0,12	1,64	35,8
		ramsey	0,29	5,18	0,21	0,07	1,49	0,63	1,82	11,42
		random-net	0,07	1,4	0,2	>0,01	8,48	>0,01	18,1	4,02
		set-covering	0,89	0,27	0,7	0,02	0,58	0,02	0,35	0,6
		wmaxcut	0,08	13,74	0,05	0,44	12,65	6,21	8,56	22,48
	random	wmax2sat	0,09	3,35	0,03	21,57	0,47	11,33	2,92	40,64
		wmax3sat	0,89	2,02	0,97	3,11	3,31	1,43	5,96	38,7
		wpmax2sat	0,44	5,21	0,08	12,91	0,67	6,76	2,71	32,77
		wpmax3sat/hi	0,81	1,12	0,48	1,88	1,28	1,17	2,38	33,9
Ensemble			0,5	4,43	0,29	6,84	3,24	8,13	2,54	31,09

6.3 Évaluation empirique de l'apprentissage des UCS

Nous présentons dans cette section une évaluation expérimentale de l'impact sur les performances de notre solveur AHMAXSAT de l'ajout des motifs des k-UCS à son schéma d'apprentissage. Tous les tests sont effectués sur le benchmark et selon le protocole expérimental décrits en section 2.6.

En partant d'une variante d'AHMAXSAT utilisant les motifs du schéma d'apprentissage classique ($\{2, 3^b\}$-UCS), nous ajoutons successivement les motifs de l'ensemble 3^t-UCS, puis ceux de 4-UCS et ceux de 5-UCS. Des résultats préliminaires (non présentés dans ce chapitre) suggèrent que l'ajout des motifs des ensembles k-UCS avec $k > 5$ ont un impact négatif sur les performances du solveur. Toutes les variantes incluent également dans leur schéma d'apprentissage le motif 2.17 qui n'est pas un UCS. Les résultats sont présentés dans les tables 6.2 et 6.3. Pour chaque variante d'AHMAXSAT, nous présentons le nombre d'instances résolues (colonnes T), le nombre moyen de nœuds explorés (colonnes D), le temps moyen de résolution (colonnes T), le nombre moyen de propagations effectuées à chaque nœud de l'arbre de recherche (colonnes P/D) et le nombre moyen de sous-ensembles inconsistants détectés à chaque nœuds de l'arbre de recherche (colonnes □/D). Le rôle de ces deux derniers indicateurs est de mesurer la réduction des redondances dans l'application de la propagation unitaire et dans la détection des sous-ensembles inconsistants. Ils peuvent également indiquer une perte dans la précision de l'estimation de la borne inférieure. La figure 6.2 compare les temps de résolution instance par instance pour chaque couple de variantes considéré dans la suite de cette section.

$\{2, 3^t\}$-UCS vs. $\{2, 3\}$-UCS Comme on pouvait s'y attendre, l'ajout des 3^t-UCS aux motifs utilisés par les solveurs existants (les $\{2, 3^b\}$-UCS) ne change pas fondamentalement le comportement d'AHMAXSAT puisque les sous-ensembles inconsistants ne correspondent que très rarement à ces motifs (0,29% en moyenne, cf. table 6.1). Sur les classes d'instances où le taux d'apparition des 3^t-UCS n'est pas nul, on peut observer une légère réduction du nombre moyen de propagations et de sous-ensembles inconsistants détectés par nœud de l'arbre de recherche (colonnes P/D et □/D) qui se traduit par une légère amélioration de la vitesse d'exploration de l'arbre de recherche (non représentée dans la table 6.2). Le nombre moyen de nœuds explorés et le temps moyen de résolution sont aussi réduits sur ces classes d'instances (colonnes D et T).

Par exemple, sur la classe ms/random/max3sat, qui a le plus fort pourcentage d'apparition des 3^t-UCS dans les sous-ensembles inconsistants (1,45% en moyenne), on peut voir que le nombre moyen de propagations et de sous-ensembles inconsistants détectés par nœud sont réduits respectivement de 1% et 2%. La vitesse moyenne d'exploration de l'arbre de recherche passe de 1306,5 nœuds par secondes à 1326,5 (+1,5%). De plus, le nombre de nœuds explorés est réduit de 5,1%. La combinaison de ces deux effets produit une réduction du temps moyen de résolution de 6,1%.

Ces résultats confirment les bénéfices attendus de l'apprentissage des transformations des UCS :

1. l'augmentation de la vitesse d'exploration de l'arbre de recherche due à la réduction de la redondance dans le calcul de la borne inférieure, et
2. la réduction du nombre de nœuds explorés due à l'amélioration de la qualité de la borne inférieure induite par l'augmentation du nombre de clauses unitaires.

TABLE 6.2 – Comparaison de l'impact des schémas d'apprentissage basés sur les motifs $\{2, 3^b\}$-UCS, $\{2, 3\}$-UCS et $\{2, 3, 4\}$-UCS. Les deux premières colonnes montrent les classes d'instances et leurs tailles. Les colonnes S, D, T, P/D et □/D donnent respectivement le nombre d'instances résolues et les moyennes de : nombre de nœuds explorés, temps d'exécution, nombre de propagations et de sous-ensembles inconsistants détectés par nœud. Les colonnes marquées d'une étoile ne considèrent que les instances résolues par toutes les variantes.

		Classes d'instances	#	$\{2,3^b\}$-UCS					$\{2,3\}$-UCS					$\{2,3,4\}$-UCS				
				S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*
ms	crafted	bipartite	100	100	33658	88,26	1135,29	73,95	100	33726	88,37	1135,40	73,95	100	33485	87,85	1135,56	73,95
		maxcut	67	56	241129	63,14	146,12	21,97	56	241129	63,03	146,12	21,97	56	164939	42,08	109,17	15,49
		set-covering	10	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	highgirth	82	7	4563195	1045,30	43,15	2,50	7	4649532	1064,17	43,12	2,49	7	4636028	1065,87	43,14	2,49
		max2sat	100	100	30406	62,12	926,92	48,64	100	30484	62,37	927,52	48,72	93	174399	291,28	699,58	31,64
		max3sat	100	100	372213	284,85	231,57	24,49	100	353221	267,49	229,89	24,00	100	355812	250,40	213,11	21,11
		min2sat	96	96	1067	2,66	1097,45	35,87	96	1067	2,66	1097,92	35,88	96	1259	2,46	843,00	19,87
sms	crafted	frb	25	5	1364465	333,46	49,81	6,91	5	1364465	333,80	49,81	6,91	5	801715	205,16	47,11	5,20
		job-shop	3	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
		maxclique	158	133	25474	28,70	88,35	7,48	133	25474	27,84	88,35	7,48	133	22426	30,48	86,16	6,40
		maxone	140	109	148070	31,20	718,78	23,17	109	148071	31,41	718,59	23,16	107	133744	28,46	717,18	22,99
		min-enc/kbtree	42	34	202579	500,82	1035,30	63,69	34	208641	515,83	1035,10	63,69	34	202738	488,35	1027,48	62,76
		pseudo/miplib	4	2	268	0,02	11,81	1,76	2	268	0,03	11,81	1,76	2	268	0,03	11,81	1,76
		reversi	44	8	4679	129,95	2596,21	23,86	8	4679	129,30	2596,21	23,86	8	4719	129,99	2648,11	24,46
		scheduling	5	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	min2sat	60	58	5271	68,74	6055,45	51,88	58	5278	68,62	6055,49	51,88	58	5281	69,40	6056,51	51,88
		min3sat	60	58	67532	181,22	1199,20	27,60	58	67587	181,62	1199,31	27,60	58	66974	180,66	1195,63	27,32
		pmax2sat	60	60	1045	3,49	944,41	74,60	60	1066	3,56	946,50	74,55	60	2781	8,86	863,36	62,01
		pmax3sat/hi	30	30	89897	65,05	242,44	16,43	30	89236	64,56	242,18	16,37	30	86400	60,90	236,21	15,61
wpms	crafted	auction	40	40	467409	123,48	143,71	66,41	40	467518	124,13	143,38	66,26	40	309981	111,12	127,15	73,94
		CSG	10	4	54378	432,44	16,91	0,03	4	54378	453,03	16,91	0,03	4	54648	432,03	15,44	0,03
		frb	34	14	492276	128,57	40,37	6,01	14	492276	125,02	40,37	6,01	14	290082	82,42	36,08	4,43
		min-enc	74	64	21340	109,06	8805,10	78,21	64	21340	109,17	8805,10	78,21	64	24009	117,80	8561,59	73,47
		pseudo/miplib	12	4	1500	159,40	2858,98	162,77	4	1500	159,33	2858,98	162,77	4	1438	155,80	2602,51	166,43
		ramsey	15	4	158529	44,17	6,97	2,14	4	158529	44,30	6,97	2,14	4	158131	44,33	7,01	2,15
		random-net	32	1	1253990	1069,46	-	-	1	1253990	1078,59	-	-	1	1563328	1366,22	-	-
		set-covering	45	25	3754	42,31	453,72	46,78	25	3754	42,47	453,72	46,78	25	3754	42,74	453,72	46,78
		wmaxcut	48	46	53152	85,89	597,06	122,46	46	53214	86,56	597,13	122,46	46	28518	51,44	434,52	97,32
	random	wmax2sat	120	120	4192	48,07	2595,47	279,33	120	4188	47,81	2596,14	279,54	119	13014	131,60	2307,88	181,96
		wmax3sat	40	40	41332	115,25	612,39	106,68	40	40387	112,18	611,15	105,80	40	38980	106,98	598,19	100,12
		wpmax2sat	90	90	575	5,87	1172,31	209,22	90	573	5,74	1170,66	210,00	90	841	12,03	1322,33	209,33
		wpmax3sat/hi	30	30	31591	88,38	656,63	64,58	30	31464	87,78	656,84	64,51	30	30751	85,57	653,73	63,34
		Ensemble	1776	1438	112580	92,74	1344,68	77,37	1438	111719	91,85	1344,62	77,37	1428	108233	109,26	1273,27	64,24

TABLE 6.3 – Comparaison de l'impact des schémas d'apprentissage basés sur les motifs $\{2,3,4^t\}$-UCS, $\{2,3,4^t,5\}$-UCS et $\{2,3,4^t,5^t\}$-UCS. Les deux premières colonnes montrent les classes d'instances et leurs tailles. Les colonnes S, D, T, P/D et □/D donnent respectivement le nombre d'instances résolues et les moyennes de : nombre de nœuds explorés, temps d'exécution, nombre de propagations et de sous-ensembles inconsistants détectés par nœud. Les colonnes marquées d'une étoile ne considèrent que les instances résolues par toutes les variantes.

Classes d'instances		#	$\{2,3,4^t\}$-UCS					$\{2,3,4^t,5\}$-UCS					$\{2,3,4^t,5^t\}$-UCS				
			S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*	S	D*	T*	P/D*	□/D*
ms	crafted	bipartite	100	100	33518	90,68	1135,76	73,96	98	184518	318,89	687,84	30,05	100	32328	84,04	1132,43	73,41
		maxcut	67	56	167956	48,47	142,85	20,09	56	192076	49,04	127,83	17,27	56	171380	44,72	135,16	18,32
		set-covering	10	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	highgirth	82	7	4647438	1121,22	43,04	2,49	6	4680189	1086,23	43,09	2,49	7	4650016	1072,92	43,25	2,50
		max2sat	100	100	29710	62,39	926,29	48,58	90	206837	334,29	681,04	29,77	100	28292	55,55	890,84	44,43
		max3sat	100	100	325667	242,30	221,25	22,34	100	367432	249,26	202,37	19,65	100	340968	231,18	205,83	20,07
		min2sat	96	96	1065	2,70	1096,26	35,78	96	3689	7,02	844,63	20,86	96	1046	2,56	1096,05	35,75
pms	crafted	frb	25	5	804408	209,33	46,98	5,19	5	687551	176,42	43,94	4,64	5	585949	154,16	46,13	4,79
		job-shop	3	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
		maxclique	158	133	22340	28,35	86,13	6,39	133	26735	33,89	231,73	5,81	133	23107	30,24	86,08	6,11
		maxone	140	109	133749	28,76	718,57	23,14	108	133742	28,25	1105,50	27,39	109	133747	28,70	718,69	23,12
		min-enc/kbtree	42	34	209900	507,22	1032,18	63,54	35	205645	493,96	979,42	60,82	34	197998	476,23	1036,96	63,45
		pseudo/miplib	4	2	268	0,02	11,81	1,76	2	288	0,03	4726,24	17,03	2	288	0,02	11,81	1,76
		reversi	44	8	4736	128,09	2637,67	24,47	8	4708	132,24	5135,31	34,63	8	4555	129,24	2907,36	26,98
		scheduling	5	0	-	-	-	-	0	-	-	-	-	0	-	-	-	-
	random	min2sat	60	58	5283	69,24	6056,48	51,88	51	12161	161,88	5658,33	42,06	58	5283	69,44	6056,44	51,88
		min3sat	60	58	66887	186,22	1195,86	27,31	52	187338	451,47	1018,56	23,05	58	66917	181,23	1195,76	27,31
		pmax2sat	60	60	1037	3,53	931,43	72,41	60	2257	6,77	838,20	58,57	60	1021	3,18	878,41	65,75
		pmax3sat/hi	30	30	86932	62,22	240,40	16,11	30	86893	60,28	229,83	17,36	30	84944	60,08	235,81	15,59
wpms	crafted	auction	40	40	310614	111,95	126,81	73,73	40	310322	118,37	111,79	72,26	40	310021	115,49	116,66	74,61
		CSG	10	4	54378	445,37	16,91	0,03	4	54813	431,69	14,24	1,20	4	54813	452,34	16,43	0,03
		frb	34	14	291044	78,69	36,03	4,42	14	248789	71,45	35,40	4,14	14	212369	58,74	36,49	4,27
		min-enc	74	64	20522	107,61	8805,42	76,83	64	21554	111,64	10838,84	74,83	64	20101	108,83	8826,32	77,08
		pseudo/miplib	12	4	1493	157,57	2906,18	169,18	4	1496	159,45	158,51	32,86	4	1505	158,77	2852,65	166,42
		ramsey	15	4	158161	44,64	7,01	2,15	4	157496	44,98	23,32	7,21	4	157559	44,79	7,04	2,16
		random-net	32	1	1563328	1360,23	-	-	3	335161	239,96	-	-	3	335161	237,55	-	-
		set-covering	45	25	3754	44,07	453,72	46,78	25	3764	42,95	501,70	51,02	25	3754	46,41	453,72	46,78
		wmaxcut	48	46	28679	53,70	599,89	109,61	46	28737	51,11	613,36	99,55	46	26985	49,41	582,41	97,91
	random	wmax2sat	120	120	4182	48,05	2591,59	277,57	118	16188	170,24	2226,11	169,67	120	3810	43,18	2538,31	261,95
		wmax3sat	40	40	37939	106,24	605,06	102,71	40	38221	103,03	579,57	95,29	40	36836	99,52	591,31	97,66
		wpmax2sat	90	90	575	5,68	1156,08	211,80	90	777	10,48	1270,58	197,03	90	575	5,52	1144,55	205,50
		wpmax3sat/hi	30	30	31004	87,47	656,15	64,11	30	30705	85,05	660,87	64,44	30	30451	84,09	654,01	63,38
Ensemble		1776	1438	96202	87,31	1340,69	76,51	1412	126692	144,32	1347,10	58,56	1440	94407	82,66	1328,76	73,28

On peut noter également que ce second effet masque en partie l'impact du premier dans les statistiques que nous présentons. La réduction de la redondance fait baisser les valeurs des indicateurs P/D et □/D tandis qu'à l'inverse, l'augmentation du nombre de clauses unitaires les augmente.

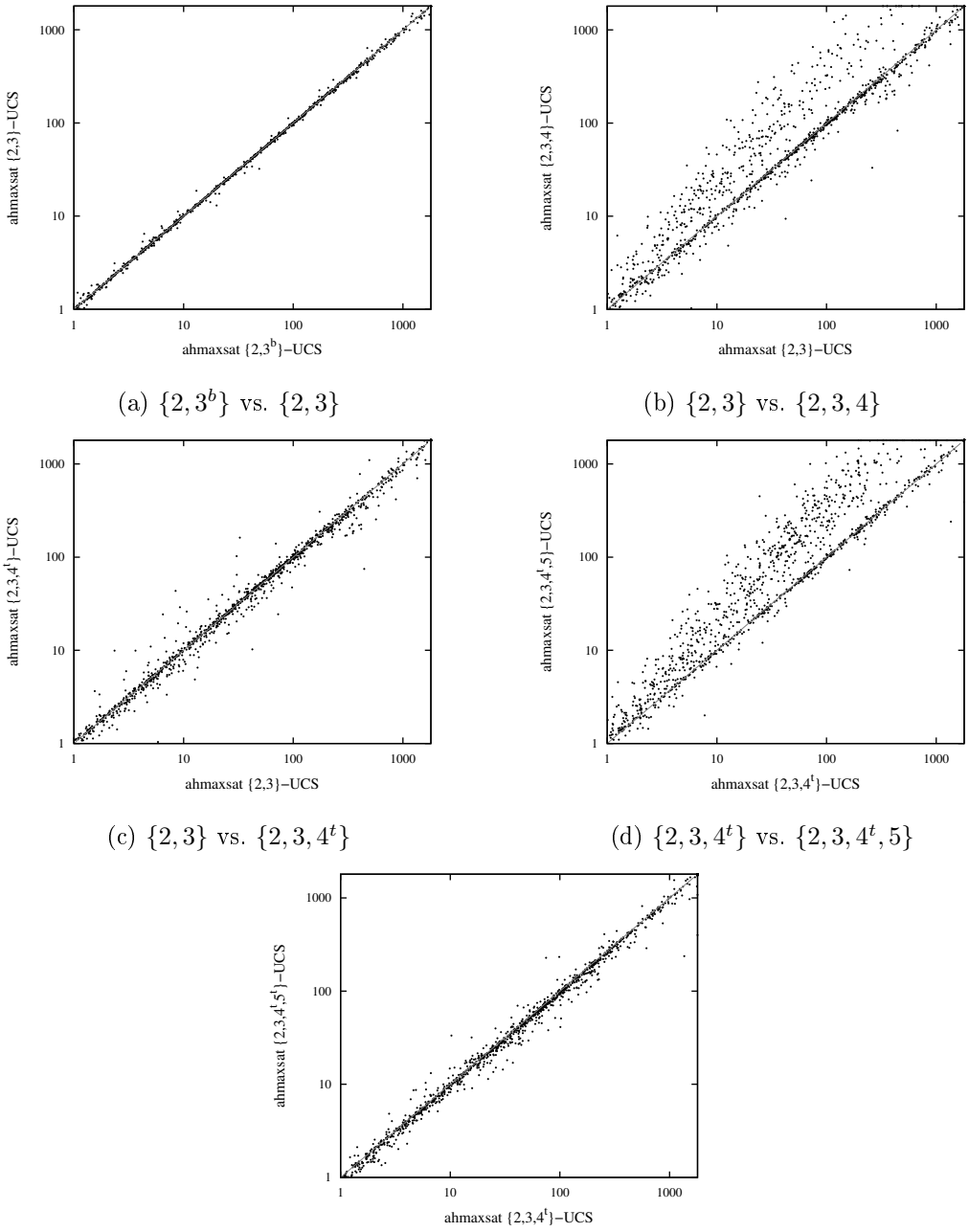
Si l'on considère les résultats sur l'ensemble du benchmark, le nombre moyen de nœuds explorés et le temps moyen de résolution sont réduits respectivement de 0,8% et 1%. Comme nous l'avons dit précédemment, la faible ampleur de ces résultats est à mettre en rapport avec la faible fréquence d'apparitions des motifs de l'ensemble 3^t-UCS dans les sous-ensembles inconsistants détectés par AHMAXSAT.

$\{2,3\}$-UCS vs. $\{2,3,4\}$-UCS Si l'on ajoute les motifs de l'ensemble 4-UCS au schéma d'apprentissage, les nombres moyens de propagations et de sous-ensembles inconsistants détectés par décisions sont réduits significativement (respectivement -5,3% et -17,1%). Comme la quantité d'apprentissage réalisée par le solveur augmente, on peut s'attendre à une légère réduction de ces deux valeurs, mais pas dans de telles proportions. Ces valeurs indiquent probablement une perte dans la capacité de détection des sous-ensembles inconsistants, et donc une réduction de la qualité de la borne inférieure. On peut remarquer que sur les classes d'instances présentant les plus fortes réductions de ces indicateurs, comme ms/random/max2sat ou wpms/random/wmax2sat, le nombre moyen de nœuds explorés augmente fortement (respectivement +472,1% et +210,7%) ce qui vient confirmer notre interprétation. Sur l'ensemble du benchmark, 10 instances de moins sont résolues et le temps moyen de résolution est augmenté de 19,1%.

Il est intéressant d'observer que les résultats varient fortement selon les classes d'instances. La perte de performances ne survient que sur les classes d'instances ayant de forts pourcentages d'apparitions des 4^b-UCS. À l'inverse, sur les classes d'instances ayant de faibles pourcentages d'apparitions des 4^b-UCS, comme pms/crafted/frb ou wpms/crafted/frb, le temps moyen de résolution est réduit (-39,5% et -34,1% respectivement). Ces résultats suggèrent que la mémorisation des transformations des sous-ensembles inconsistants correspondant aux motifs 4^b-UCS impacte négativement et dans des proportions importantes les performances de notre solveur.

$\{2,3\}$-UCS vs. $\{2,3,4^t\}$-UCS Si l'on ignore les 4^b-UCS et que l'on ne mémorise que les transformations des sous-ensembles inconsistants correspondant aux motifs $\{2,3,4^t\}$-UCS, le nombre moyen de nœuds explorés et le temps moyen de résolution sur l'ensemble du benchmark sont réduits respectivement de 13,9% et de 4,9%. On peut observer que les nombres moyens de propagations et de sous-ensembles inconsistants détectés par nœud sont légèrement réduits (respectivement -0,3% et -1,2%).

$\{2,3,4^t\}$-UCS vs. $\{2,3,4^t,5\}$-UCS Lorsqu'on ajoute les motifs 5-UCS au schéma d'apprentissage d'AHMAXSAT, on observe le même comportement qu'avec les motifs 4-UCS. Sur l'ensemble du benchmark, le nombre moyen de sous-ensembles inconsistants détectés par nœud est réduit significativement (-23,3%). En conséquence, le nombre moyen de nœuds explorés augmente (+31,7%), 26 instances de moins sont résolues et le temps moyen de résolution est plus élevé (+65,3%). Comme précédemment, la perte de performances est particulièrement élevée sur les classes d'instances ayant un fort pourcentage d'apparitions des motifs 5^t-UCS dans les sous-ensembles inconsistants, comme les



(a) $\{2, 3^b\}$ vs. $\{2, 3\}$

(b) $\{2, 3\}$ vs. $\{2, 3, 4\}$

(c) $\{2, 3\}$ vs. $\{2, 3, 4^t\}$

(d) $\{2, 3, 4^t\}$ vs. $\{2, 3, 4^t, 5\}$

(e) $\{2, 3, 4^t\}$ vs. $\{2, 3, 4^t, 5^t\}$

FIGURE 6.2 – Comparaison détaillée des temps de résolution des variantes d'AHMAXSAT. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

ms/crafted/bipartite, ms/random/max2sat, pms/random/min3sat ou wpms/random/wmax2sat où l'augmentation du temps moyen de résolution est respectivement de 251,5%, 435,7%, 142,5% et 254,6%.

$\{2, 3, 4^t\}$-UCS vs. $\{2, 3, 4^t, 5^t\}$-UCS Comme précédemment, si l'on ignore les motifs 5^b-UCS et qu'on ne mémorise que les motifs $\{2, 3, 4^t, 5^t\}$-UCS, 2 instances de plus sont résolues, le nombre moyen de nœuds et le temps moyen de résolution sont réduits respectivement de 1,9% et 5,3%.

Bilan L'ajout des motifs $\{3^t, 4^t, 5^t\}$-UCS au schéma d'apprentissage d'AHMAXSAT permet de résoudre 2 instances de plus. Le nombre moyen de nœuds explorés et le temps moyen de résolution sont réduits de 16,2% et 11,8% respectivement. Les figures 6.4a et 6.4b comparent instance par instance respectivement le temps de résolution et le nombre de nœuds explorés d'AHMAXSAT$\{2, 3^b\}$-UCS et d'AHMAXSAT$\{2, 3, 4^t, 5^t\}$-UCS. On peut observer que le gain est régulièrement réparti sur l'ensemble du benchmark.

Enfin, la figure 6.3 compare les temps de résolution cumulés des différentes variantes du solveur testées dans cette section. On remarque que les performances des variantes utilisant les motifs $\{2, 3, 4\}$-UCS et $\{2, 3, 4^t, 5\}$-UCS sont nettement en retrait par rapport aux autres variantes. On peut voir également que les ajouts successifs des ensembles de motifs 3^t-UCS, 4^t-UCS et 5^t-UCS au schéma d'apprentissage d'AHMAXSAT permettent d'améliorer ses performances.

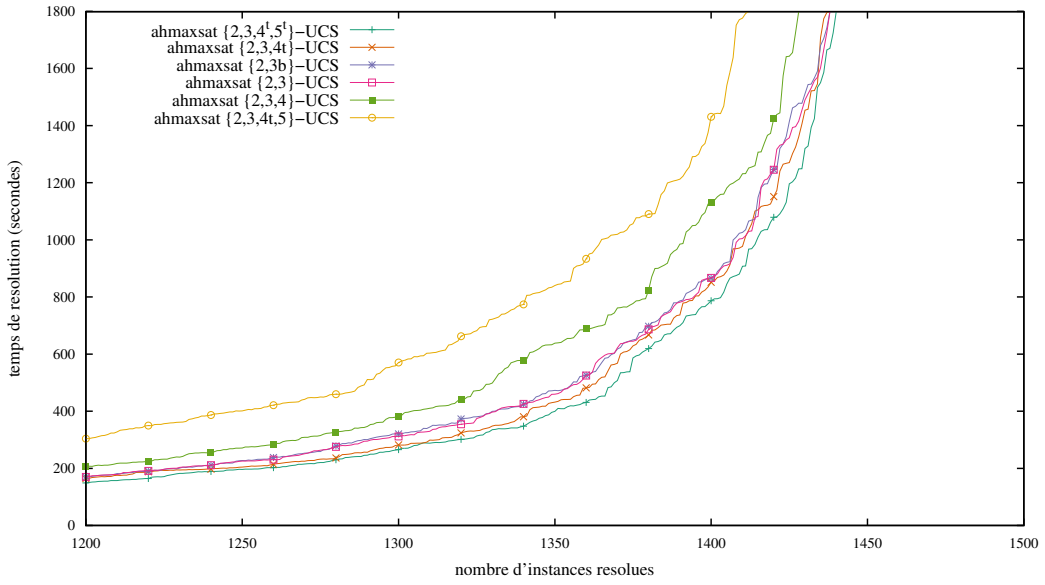
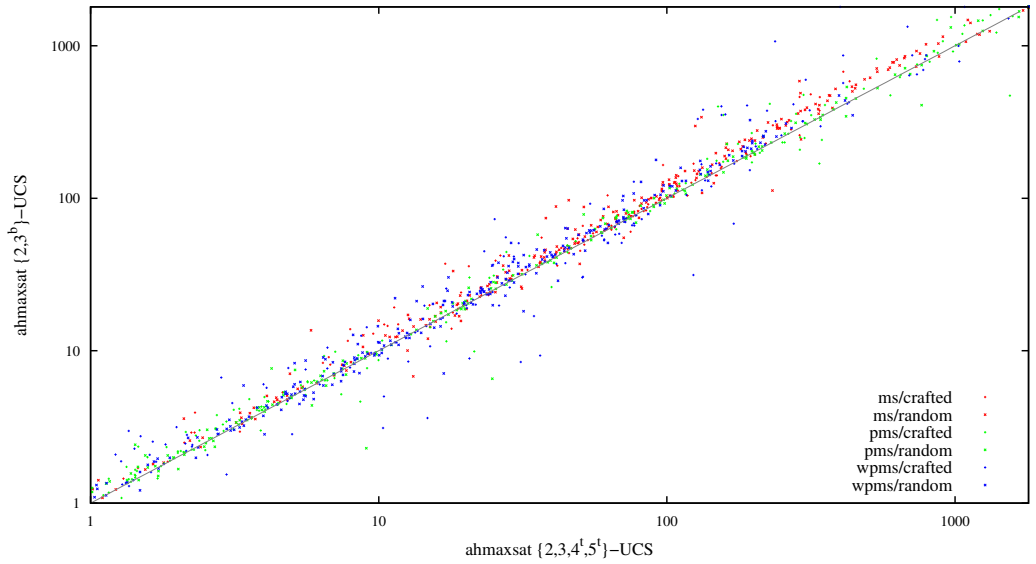
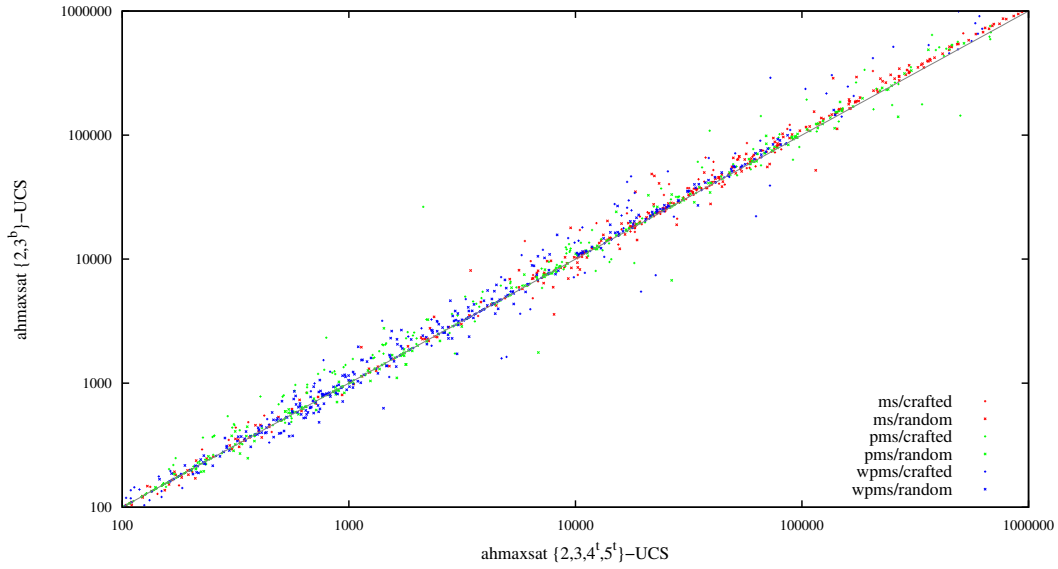


FIGURE 6.3 – Comparaison des temps d'exécution cumulés des variantes d'AHMAXSAT, avec en abscisse le nombre d'instances résolues et en ordonnée le temps de résolution.



(a) Temps de résolution (en secondes)



(b) Nombre de nœuds

FIGURE 6.4 – Comparaison du nombre de nœuds explorés et du temps de résolution d'AHMAXSAT avec les ensembles de motifs $\{2, 3^b\}$-UCS et $\{2, 3, 4^t, 5^t\}$-UCS. Chaque point représente une instance. Tous les axes sont en échelle logarithmique.

6.4 Conclusion

Nous avons présenté dans ce chapitre un ensemble de motifs permettant d'étendre l'apprentissage fait par les solveurs de type séparation et évaluation. Les transformations mémorisées grâce à ces motifs permettent de réduire la redondance dans le calcul de la borne inférieure, et donc d'améliorer la vitesse d'exploration de l'arbre de recherche. Elles génèrent également des clauses unitaires qui permettent de détecter plus de sous-ensembles inconsistants et donc d'obtenir des estimations de la borne inférieure de meilleure qualité.

L'étude expérimentale que nous avons menée confirme l'intérêt de ces motifs et montre qu'ils permettent d'améliorer significativement les performances de notre solveur AHMAXSAT. Les résultats obtenus montrent également que l'inclusion de certains motifs (les 4^b-UCS et 5^b-UCS) au schéma d'apprentissage réduit considérablement la qualité de l'estimation de la borne inférieure. Nous étudierons ce phénomène dans le prochain chapitre de cette thèse.

Chapitre 7

UP-résilience

Nous avons vu dans le chapitre précédent que l'ajout des ensembles de motifs 4^b-UCS et 5^b-UCS aux critères d'apprentissage d'AHMAXSAT détériorait significativement ses performances. En particulier, les nombres moyens de propagations faites et de sous-ensembles inconsistants détectés à chaque nœud de l'arbre de recherche étaient significativement réduits lorsque les transformations de ces motifs étaient mémorisées. Ces résultats suggèrent que les transformations par max-résolution, et à plus forte raison quand elles sont mémorisées dans la sous-partie de l'arbre de recherche, affectent l'efficacité de la propagation unitaire et par là même sa capacité à détecter les sous-ensembles inconsistants.

Nous étudions dans ce chapitre ce phénomène. Nous montrons que dans certains cas les informations pouvant mener à la propagation de variables sont, après transformation par max-résolution, fragmentées en plusieurs clauses. La propagation unitaire est alors moins efficace dans la formule transformée que dans l'originale. Nous introduisons la notion d'UP-résilience d'une transformation, qui caractérise les transformations n'ayant pas d'impact sur la propagation unitaire et qui, plus généralement, permet de quantifier cet impact. Nous montrons que, selon ce critère, les transformations des sous-ensembles de la formule qui corespondent aux motifs utilisés comme critères d'apprentissage n'altèrent pas l'efficacité de la propagation unitaire. Ceci contribue à expliquer d'un point

de vue théorique les résultats empiriques obtenus ces dix dernières années sur le développement des règles d'inférence et de l'apprentissage pour les solveurs de type séparation et évaluation. Nous étudions empiriquement l'effet de l'ordre d'application des étapes de max-résolution (cf. chapitre 4) sur l'UP-résilience des transformations et nous évaluons les critères d'apprentissage (cf. chapitre 6) au regard de l'UP-résilience. Les résultats obtenus permettent de mieux comprendre le fonctionnement du mécanisme d'apprentissage et ouvrent de nouvelles perspectives de développement.

Ce chapitre est organisé comme suit. Nous rappelons dans la première partie les règles de transformation et d'apprentissage utilisées par les solveurs existants. La seconde partie est consacrée à la notion d'UP-résilience. Nous donnons sa définition formelle et discutons des propriétés de ces transformations. Dans la troisième et dernière partie, nous présentons et analysons les résultats de l'étude expérimentale que nous avons menée. Les travaux et résultats présentés dans ce chapitre ont été précédemment publiés [AH15c].

7.1 Préliminaires et motivations

Nous rappelons brièvement dans cette section les mécanismes de détection et de transformation des sous-ensembles inconsistants (IS) appliqués par les solveurs de type séparation et évaluation durant le calcul de la borne inférieure. Nous présentons les critères d'apprentissages existant avant d'illustrer par un exemple l'impact que peut avoir les transformations des sous-ensembles inconsistants sur l'efficacité de la propagation unitaire.

7.1.1 Détection et transformation des sous-ensembles inconsistants

La propagation unitaire (*unit propagation*, UP), présentée dans la section 1.2.3.2 de cette thèse, est une règle d'inférence sémantique qui consiste à satisfaire itérativement les littéraux apparaissant dans des clauses unitaires jusqu'à ce qu'un conflit soit détecté (i.e. une clause vide) ou qu'il n'y ait plus aucune clause unitaire. Lorsqu'une clause vide est détectée, l'ensemble des clauses qui ont menées, par propagation unitaire, à falsifier ses littéraux forment un sous-ensemble inconsistant de la formule.

Comme nous l'avons vu dans la section 2.3.4, cette méthode est utilisée (dans la propagation unitaire simulée, SUP, et la méthode des littéraux contradictoires, FL) par les solveurs de type séparation et évaluation lors du calcul de la borne inférieure pour détecter les sous-ensembles inconsistants. De la capacité de la propagation unitaire à détecter les sous-ensembles inconsistants dépend la précision de l'estimation de la borne inférieure, et donc la capacité des solveurs à faire des coupes dans l'arbre de recherche.

Une fois détectés, les sous-ensembles inconsistants doivent être transformés pour s'assurer qu'ils ne soient comptés qu'une fois. Une des méthodes de transformation existante est basée sur la règle de la max-résolution (cf. section 2.2.3 et 2.3.4). Nous la rappelons ci-dessous.

Définition 38 (Transformation par max-resolution)**.** Soit Φ une formule et ψ un sous-ensemble inconsistant de Φ. Pour toute variable x_i telle que $\exists! c_j \in \psi$ t.q. $x_i \in c_j$ et $\exists! c_k \in \psi$ t.q. $\overline{x_i} \in c_k$, nous notons $\theta(\psi, x_i)$ l'ensemble des clauses obtenues après l'application de la max-résolution entre $c_j = \{x_i, l_{j_1}, \ldots, l_{j_s}\}$ et $c_k = \{\overline{x_i}, l_{k_1}, \ldots, l_{k_t}\}$ sur x_i.

Formellement :

$$\theta(\psi, x_i) \;=\; (\psi \setminus \{c_j, c_k\}) \cup \{cr, cc_1, \ldots, cc_{t+s}\}$$

avec $cr = \{x_{j_1}, \ldots, x_{j_s}, x_{k_1}, \ldots, x_{k_t}\}$ la clause résolvante et $cc_1, \ldots, cc_{t+s}$ les clauses de compensation définies comme :

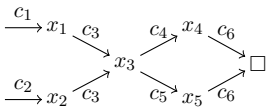
$$\begin{aligned}
cc_1 &= \{x_i, x_{j_1}, \ldots, x_{j_s}, \overline{x_{k_1}}, x_{k_2}, \ldots, x_{k_t}\},\\
cc_2 &= \{x_i, x_{j_1}, \ldots, x_{j_s}, \overline{x_{k_2}}, x_{k_3}, \ldots, x_{k_t}\},\\
&\ldots\\
cc_t &= \{x_i, x_{j_1}, \ldots, x_{j_s}, \overline{x_{k_t}}\},\\
cc_{t+1} &= \{\overline{x_i}, x_{k_1}, \ldots, x_{k_t}, \overline{x_{j_1}}, x_{j_2}, \ldots, x_{j_s}\},\\
cc_{t+2} &= \{\overline{x_i}, x_{k_1}, \ldots, x_{k_t}, \overline{x_{j_2}}, x_{j_3}, \ldots, x_{j_s}\},\\
&\ldots\\
cc_{t+s} &= \{\overline{x_i}, x_{k_1}, \ldots, x_{k_t}, \overline{x_{j_s}}\}
\end{aligned}$$

Pour n'importe quelle séquence $S = (x_{i_1}, \ldots, x_{i_k})$ de variables apparaissant dans ψ, nous notons $\Theta(\psi, S)$ l'ensemble des clauses obtenues après l'application de la max-résolution sur x_{i_1} puis x_{i_2} et ainsi de suite. Formellement :

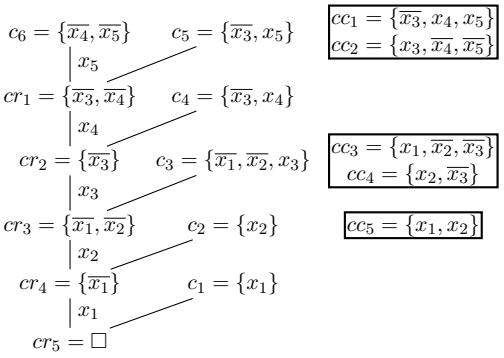
$$\Theta(\psi, S) \;=\; \theta(\ldots\theta(\theta(\psi, x_{i_1}), x_{i_2})\ldots, x_{i_k})$$

L'exemple suivant illustre cette méthode de transformation.

Exemple 19. Considérons la formule CNF non valuée $\Phi_1 = \{c_1, \ldots, c_6\}$ avec $c_1 = \{x_1\}$, $c_2 = \{x_2\}$, $c_3 = \{\overline{x_1}, \overline{x_2}, x_3\}$, $c_4 = \{\overline{x_3}, x_4\}$, $c_5 = \{\overline{x_3}, x_5\}$ et $c_6 = \{\overline{x_4}, \overline{x_5}\}$. L'application de SUP* sur Φ_1 conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4, x_5@c_5\rangle$ (x_1 est propagé par la clause c_1, puis x_2 par c_2 et ainsi de suite). La clause c_6 est vide. Le graphe d'implications décrivant ces étapes de propagation unitaire est présenté dans la figure 7.1a. Φ_1 est donc inconsistante. Sa transformation par max-résolution est faite comme suit. La max-résolution est appliquée entre les clauses c_5 et c_6 autour de la variable x_5. La clause résolvante intermédiaire $cr_1 = \{\overline{x_3}, \overline{x_4}\}$ est produite ainsi que les



(a) Graphe d'implication



(b) Étapes de max-résolution

FIGURE 7.1 – Graphe d'implications et étapes de max-résolution illustrant l'exemple 19.

clauses de compensation $cc_1 = \{\overline{x_3}, x_4, x_5\}$ et $cc_2 = \{x_3, \overline{x_4}, \overline{x_5}\}$. Les clauses originales c_5 et c_6 sont supprimées de la formule. Puis, la max-résolution est appliquée entre le résolvant intermédiaire cr_1 et la clause originale suivante c_4 sur la variable x_4 et ainsi de suite. La figure 7.1b montre les étapes de max-résolution appliquées, avec les clauses de compensation encadrées. Après transformation complète, nous obtenons la formule $\Phi'_1 = \{\square, cc_1, cc_2, cc_3, cc_4, cc_5\}$ avec $cc_3 = \{x_1, \overline{x_2}, \overline{x_3}\}$, $cc_4 = \{x_2, \overline{x_3}\}$ et $cc_5 = \{x_1, x_2\}$.

7.1.2 Schémas d'apprentissage

Les solveurs de type séparation et évaluation récents appliquent une forme limitée d'apprentissage en conservant dans la sous-partie de l'arbre de recherche les transformations des sous-ensembles inconsistants par max-résolution. Cet apprentissage n'est réalisé que lorsque les sous-ensembles inconsistants remplissent certaines conditions : les critères d'apprentissage.

Comme nous l'avons vu précédemment (cf. section 2.3.4.3), deux schémas d'apprentissage existent. Le premier est basé sur la taille des résolvants intermédiaires. Les transformations par max-résolution ne sont conservées que si tous les résolvants intermédiaires sont de tailles strictement inférieures à trois. Le second type de schéma d'apprentissage est basé sur une ensemble de motifs. Ne sont conservées que les transformations des parties des sous-ensembles inconsistants qui correspondent à ces motifs. Les principaux motifs utilisés par les solveurs existants sont les suivants (avec en haut les clauses originales des sous-ensembles inconsistants et en bas les clauses obtenues après transformation) :

$$\frac{\{\{l_1, l_2\}, \{l_1, \overline{l_2}\}\}}{\{\{l_1\}\}} \tag{2.15}$$

$$\frac{\{\{l_1, l_2\}, \{l_1, l_3\}, \{\overline{l_2}, \overline{l_3}\}\}}{\{\{l_1\}, \{l_1, l_2, l_3\}, \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}\}} \tag{2.16}$$

$$\frac{\{\{l_1\}, \{\overline{l_1}, l_2\}, \{\overline{l_2}, l_3\}, \ldots, \{\overline{l_{k-1}}, l_k\}, \{\overline{l_k}\}\}}{\{\square, \{l_1, \overline{l_2}\}, \{l_2, \overline{l_3}\}, \ldots, \{l_{k-1}, \overline{l_k}\}\}} \tag{2.17}$$

Dans le chapitre précédent, nous avons introduit de nouveaux ensembles de motifs et montré expérimentalement leur apports dans notre solveur AHMAXSAT. Dans la suite de ce chapitre, nous distinguerons les motifs (ou combinaisons de motifs) permettant la mémorisation des transformations de sous-ensembles inconsistants complets (qui produisent donc des clauses résolvantes vides) et ceux permettant la mémorisation d'UCS (qui produisent des clauses résolvantes unitaires, cf. définition 37).

7.1.3 Phénomène de fragmentation

Deux raisons sont généralement invoquées pour expliquer l'efficacité des schémas d'apprentissage existants :

— ils contrôlent l'augmentation de la taille de la formule et
— ils produisent de petites clauses de compensation qui sont plus à même d'être utilisées dans la propagation unitaire.

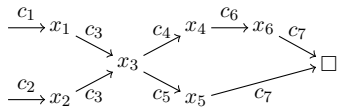
Cependant, les résultats de l'étude expérimentale présentés dans le chapitre précédent montrent qu'en ajoutant certains motifs au schéma d'apprentissage, les performances

d'AHMAXSAT étaient considérablement dégradées. Les raisons invoquées précédemment ne permettent pas d'expliquer ce comportement.

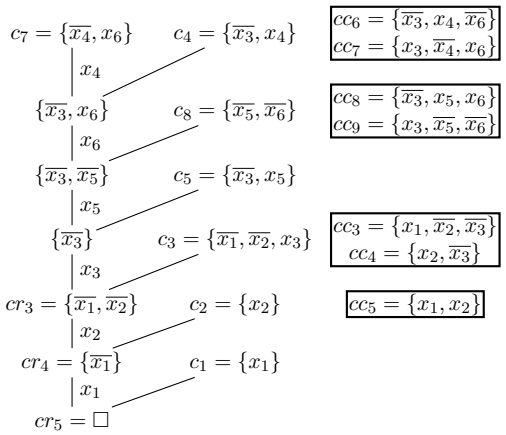
Nous illustrons dans l'exemple suivant un phénomène pouvant expliquer ces résultats. Les informations utilisables par la propagation unitaire dans la formule originale sont, après transformation par max-résolution, divisées (ou "fragmentées") dans plusieurs clauses de compensation. Elles ne sont donc pas directement utilisables par la propagation unitaire et une ou plusieurs étapes de max-résolution sont nécessaires pour les rendre exploitables.

Exemple 20. Considérons la formule CNF non valuée $\Phi_2 = \{c_1, c_2, c_3, c_4, c_5, c_7, c_8\}$ avec $c_7 = \{\overline{x_4}, x_6\}$ et $c_8 = \{\overline{x_5}, \overline{x_6}\}$ (les clauses c_1 à c_5 sont les mêmes que dans l'exemple 19). L'application de SUP* conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4, x_5@c_5, x_6@c_6\rangle$ et la clause c_7 est vide. Le graphe d'implications décrivant cette séquence est présenté dans la figure 7.2a. S'il l'on applique la max-résolution selon la séquence $\langle x_4, x_6, x_5, x_3, x_2, x_1\rangle$, on obtient la formule $\Phi'_2 = \{\square, cc_3, cc_4, cc_5, cc_6, cc_7, cc_8, cc_9\}$ avec $cc_6 = \{\overline{x_3}, x_4, \overline{x_6}\}$, $cc_7 = \{x_3, \overline{x_4}, x_6\}$, $cc_8 = \{\overline{x_3}, x_5, x_6\}$ et $cc_9 = \{x_3, \overline{x_5}, \overline{x_6}\}$ (les clauses de compensation cc_3, cc_4 et cc_5 sont les mêmes que dans l'exemple 19). La figure 7.2b montre les étapes de max-résolution réalisées.

Supposons que l'on affecte aux variables x_4 et x_5 la valeur *vrai*. Dans la formule originale Φ_2, le sous-ensemble $\{c_6, c_7\}$ est inconsistant et l'application de SUP* conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3\rangle$. Dans la formule transformée Φ'_2, les clauses cc_7 et cc_9 sont réduites et deviennent respectivement $\{x_3, x_6\}$ et $\{x_3, \overline{x_6}\}$ mais aucune étape de propagation unitaire ne peut être réalisée. On peut noter que si on appliquait la max-résolution entre ces deux clauses sur la variable x_6, on obtiendrait la clause résolvante unitaire $\{x_3\}$. Mais l'application de la propagation unitaire seule ne permet pas d'exploiter ces deux clauses. On peut donc dire que les informations qui pourraient mener à la propagation de x_3 sont fragmentées dans plusieurs clauses de compensation.



(a) Graphe d'implication



(b) Étapes de max-résolution

FIGURE 7.2 – Graphe d'implications et étapes de max-résolution illustrant l'exemple 20.

Quand de telles transformations sont apprises, cela peut affecter la capacité de la propagation unitaire à détecter des sous-ensembles inconsistants dans les nœuds de la

sous-partie de l'arbre de recherche. Par conséquent, l'estimation de la borne inférieure peut être moins précise et les solveurs peuvent être conduits à explorer davantage de nœuds.

7.2 UP-résilience

Nous étudions dans cette section la résilience de la propagation unitaire aux transformations par max-résolution. Nous introduisons la notion de transformation *UP-résiliente* qui caractérise les transformations qui n'affectent pas l'efficacité de la propagation unitaire et, plus généralement, qui permet de quantifier leur impact. Nous discutons du comportement de la propagation unitaire dans les formules transformées et montrons que les transformations des sous-ensembles inconsistants correspondant aux motifs des schémas d'apprentissage classiques n'altèrent pas l'efficacité de la propagation unitaire.

7.2.1 Dans un graphe d'implications

Quand le phénomène de fragmentation se produit, les clauses de compensation qui auraient pu propager un littéral l_i apparaissant dans un graphe d'implications G contiennent d'autres littéraux qui ne sont pas des voisins de l_i dans G. Par conséquent, pour détecter si une transformation est affectée par ce phénomène de fragmentation il est possible de vérifier si les littéraux de G peuvent être propagés lorsque seuls leurs voisins dans G sont satisfaits. Si c'est le cas, nous dirons que la transformation est UP-résiliente dans G.

Définition 39 (UP-resilience dans un graphe d'implications)**.** Soit Φ une formule, ψ un sous-ensemble inconsistant détecté par les étapes de propagation unitaire décrites par le graphe d'implications $G = (V, A)$ et S une séquence des variables apparaissant dans ψ. Pour un littéral $l_i \in V$, nous dirons que la transformation $\Theta(\psi, S)$ est UP-résiliente pour l_i dans G ssi $\square \in neigh_G(l_i)$ ou l_i peut être propagé dans $\Theta(\psi, S)|_{neigh_G(l_i)}$.

On peut généraliser cette définition à tout sous-ensemble V' de V comme suit : la transformation $\Theta(\psi, S)$ est UP-résiliente pour V' dans G ssi elle est UP-résiliente $\forall l_i \in V'$.

Enfin, nous dirons que la transformation $\Theta(\psi, S)$ est UP-résiliente dans G ssi elle est UP-résiliente pour V.

On peut noter que les voisinages incluant le nœud spécial $\square$ ne sont pas des interprétations valides. Toutes les transformations sont considérées comme étant UP-résilientes pour ces littéraux.

7.2.2 Généralisation

La définition de l'UP-résilience donnée dans la section précédente dépend du voisinage des littéraux dans le graphe d'implications. Cependant, un même sous-ensemble inconsistant peut être détecté par plusieurs séquences de propagations pouvant être décrites par plusieurs graphes d'implications distincts.

Exemple 21. Considérons à nouveau la formule Φ_1 de l'exemple 19. En plus de celle présentée dans l'exemple original, deux autres séquences de propagations peuvent conduire à

la détection du sous-ensemble inconsistant tout en changeant les valeurs données aux variables propagées : $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4, \overline{x_5}@c_6\rangle$ et $\langle x_1@c_1, x_2@c_2, x_3@c_3, x_5@c_5, \overline{x_4}@c_6\rangle$. Les graphes d'implications décrivant ces séquences sont présentés dans la figure 7.3.

FIGURE 7.3 – Graphes d'implications décrivant les étapes de propagations présentées dans l'exemple 21.

Avec la définition précédente, l'UP-résilience d'une transformation peut varier selon la manière dont le sous-ensemble inconsistant a été détecté. Pour outrepasser cette limitation, nous proposons de considérer toutes les séquences de propagations pouvant mener à la détection des sous-ensembles inconsistants. Nous définissons tout d'abord l'ensemble des voisins possibles d'un littéral apparaissant dans un graphe d'implications.

Définition 40 (Voisinages possibles)**.** Soit Φ une formule et ψ un sous-ensemble inconsistant de Φ. Pour tout littéral l_i apparaissant dans les clauses de ψ, on peut définir l'ensemble des voisinages possibles comme :

$$pneigh_\psi(l_i) = \{neigh_G(l_i),\ \forall \text{ graphe d'implications } G = (V, A) \text{ de } \psi \text{ t.q. } l_i \in V\}$$

Cette définition peut être naturellement étendue à tout ensemble de littéraux $V' \subset V$ comme suit :

$$pneigh_\psi(V') = \bigcup_{l_i \in V'} \{(n_k \setminus V') : n_k \in pneigh_\psi(l_i)\}$$

Nous pouvons maintenant donner une définition générale de l'UP-résilience qui ne dépend plus des étapes de propagations ayant conduit à la détection du sous-ensemble inconsistant.

Définition 41 (UP-résilience)**.** Soit Φ une formule, ψ un sous-ensemble inconsistant de Φ et S une séquence des variables apparaissant dans ψ. Pour tout littéral l_i apparaissant dans ψ, nous dirons que la transformation $\Theta(\psi, S)$ est UP-résiliente pour l_i ssi pour tout voisinage $n_k \in pneigh_\psi(l_i)$: $\square \in n_k$ ou l_i peut être propagé dans $\Theta(\psi, S)|_{n_k}$.

Comme précédemment, on peut étendre cette définition à tout ensemble de littéraux L apparaissant dans ψ comme suit : la transformation $\Theta(\psi, S)$ est UP-résiliente pour L ssi elle est UP-résiliente $\forall l_i \in L$.

Nous dirons que la transformation $\Theta(\psi, S)$ est UP-résiliente ssi elle est UP-résiliente pour tous les littéraux apparaissant dans ψ.

De manière similaire, on peut quantifier l'impact d'une transformation sur la propagation unitaire en définissant son pourcentage d'UP-résilience comme étant le pourcentage de couple (l_i, n_k) tels que $\square \in n_k$ ou l_i peut être propagé dans $\Theta(\psi, S)|_{n_k}$.

7.2.3 Impact sur la détection des sous-ensembles inconsistants

Une des propriétés les plus intéressantes des transformations UP-résilientes est la capacité de "récupérer" les propagations qui ne sont plus nécessaires aux sous-ensembles inconsistants transformés. On peut ainsi montrer que si une transformation est UP-résiliente pour un ensemble de littéraux, on a les mêmes propriétés pour l'ensemble des littéraux que pour chacun d'entre eux individuellement.

Propriété 10. Soit Φ une formule, ψ un sous-ensemble inconsistant de ψ et S une séquence des variables apparaissant dans ψ. Pour tout ensemble de littéraux L apparaissant dans ψ, si la transformation $\Theta(\psi, S)$ est UP-résiliente pour L alors pour tout $n_k \in pneigh_\psi(L) : \square \in n_k$ ou tous les littéraux de L peuvent être propagés dans $\Theta(\psi, S)|_{n_k}$.

Démonstration. Considérons un IS ψ détecté grâce à une séquence d'étapes de propagation unitaire décrites par le graphe d'implications G et $S = \langle x_{i_1}, \ldots, x_{i_m} \rangle$ une séquence des variables de ψ. On suppose que la transformation partielle $\Theta(\psi, \langle x_{i_1}, \ldots, x_{i_h} \rangle)$ (avec $h < m$) a déjà été appliquée. On note $G|_{\langle x_{i_1}, \ldots, x_{i_h} \rangle}$ le graphe d'implications mis à jour en remplaçant les clauses résolues par les résolvantes intermédiaires produites dans G (cf. définition 36). Pour tout littéral l_{i_j} tel que $l_{i_j}, \overline{l_{i_j}} \notin \{x_{i_1}, \ldots, x_{i_h}\}$, la clause de compensation permettant d'obtenir l'UP-résilience de l_{i_j} dans G peut être obtenue (on suppose qu'elle n'a pas déjà été obtenue et que $\square \notin neigh_G(l_{i_j})$) :

1. en appliquant la max-résolution sur l_{i_j} si $\square \notin succ_{G|_{\langle x_{i_1}, \ldots, x_{i_h} \rangle}}(l_i)$ ou
2. en appliquant la max-résolution sur un successeur l_{i_k} de l_{i_j} dans G si $\square \in succ_G(l_{i_k})$.

Ces deux points peuvent être prouvés aisément en examinant les clauses de compensation obtenues lors de la transformation des clauses contenant l_{i_j} ou $\overline{l_{i_j}}$. Dans tous les autres cas, les clauses de compensation produites ne contiennent pas l_{i_j} où elles incluent des littéraux qui ne sont pas dans le voisinage de l_{i_j} dans G. On peut noter que nous ne présentons ici que les informations pertinentes pour notre démonstration. Les conditions ci-dessus sont nécessaires mais non suffisantes pour obtenir l'UP-résilience.

Nous prouvons maintenant la validité de la propriété par induction sur la taille de l'ensemble L des littéraux apparaissant dans ψ tel que $\Theta(\psi, S)$ soit UP-résiliente pour L.
Base : Si $|L| = 1$ alors la propriété est vérifiée.
Induction : On suppose par hypothèse que la propriété est valide pour tout L' t.q. $|L'| \leq n$. Considérons un ensemble de littéraux L t.q. $|L| = n$ et un graphe d'implication $G = (V, A)$ de ψ t.q. $L \in V$. Nous considérons maintenant le dernier littéral l_{i_j} de L dans la séquence S.

Si l'UP-résilience de l_{i_j} dans G a été obtenue en appliquant la max-résolution sur l_{i_j} (cas 1), alors les clauses de compensation assurant l'UP-résilience de l_{i_j} ne peuvent contenir de littéraux de $L \setminus \{l_{i_j}\}$. Donc l'interprétation $neigh_G(L)$ est suffisante pour propager l_{i_j} dans la formule transformée. On sait par hypothèse que $neigh_G(L \setminus \{l_{i_j}\})$ propage $L \setminus \{l_{i_j}\}$ dans la formule transformée donc la propriété est valide pour L dans G.

Sinon, l'UP-résilience de l_{i_j} dans G a été obtenue grâce au cas 2. Nous considérons alors le premier littéral l_{i_k} de L ayant obtenu l'UP-résilience en appliquant la max-résolution sur un de ces successeurs x_{i_o} (cas 2). On peut distinguer deux cas :

— si $neigh_{G|_{\langle x_{i_1},\ldots,x_{i_{o-1}}\rangle}}(l_{i_k}) \cap L = \emptyset$, alors l'interprétation $neigh_G(L)$ est suffisante pour propager l_{i_k} dans la formule transformée. Comme précédemment, par hypothèse on sait que $neigh_G(L \setminus \{l_{i_k}\})$ propage $L \setminus \{l_{i_k}\}$ dans la formule transformée donc la propriété est valide pour L dans G.
— sinon, $\forall l_{i_p} \in neigh_{G|_{\langle x_{i_1},\ldots,x_{i_{o-1}}\rangle}}(l_{i_k}) \cap L$ on sait que les clauses de compensation permettant d'obtenir l'UP-résilience de l_{i_p} dans G n'ont pas encore été produites. Après l'application de la max-résolution sur x_{i_o}, on obtient le graphe d'implications mis à jour $G|_{\langle x_{i_1},\ldots,x_{i_o}\rangle}$ et $\square \in neigh_{G|_{\langle x_{i_1},\ldots,x_{i_o}\rangle}}(l_{i_p})$. Si $\square \in neigh_G(l_{i_p})$ alors $\square \in neigh_G(L)$ et la propriété est valide dans G. Sinon, la transformation ne peut être UP-résiliente dans G ce qui contredit l'hypothèse initiale.

Le même raisonnement peut être conduit pour chaque graphe d'implications G de ψ. La propriété est donc valide pour tout ensemble de littéraux apparaissant dans ψ. □

Quand un sous-ensemble ψ' d'un IS ψ n'est plus nécessaire au caractère inconsistant de ψ (par exemple lorsque ψ n'est pas minimal), cette propriété garantie que l'on puisse faire les mêmes propagations dans la formule transformée que dans les clauses originales de ψ'. L'exemple suivant illustre cette situation.

Exemple 22. Considérons à nouveau la formule Φ_1 de l'exemple 19 et sa transformation UP-résiliente en Φ'_1. Supposons que l'on affecte aux variables x_4 et x_5 la valeur *vrai*.

Dans la formule originale Φ_1, la clause c_6 est falsifiée tandis que c_4 et c_5 sont satisfaites. L'application de SUP sur $\Phi_1|_{\{x_4,x_5\}}$ conduit à la séquence de propagations $\langle x_1@c_1, x_2@c_2, x_3@c_3\rangle$. Dans la formule transformée Φ'_1, l'application de SUP sur $\Phi'_1|_{\{x_4,x_5\}}$ conduit à la séquence de propagations $\langle x_3@cc_2, x_2@cc_4, x_1@cc_3\rangle$. On peut voir que les mêmes propagations sont faites dans la formule transformée que dans l'originale.

Les mêmes propagation peuvent être faites dans la formule Φ_2 de l'exemple 20 avec la même interprétation $\{x_4, x_5\}$. Mais si l'on considère la formule Φ'_2 obtenue par la transformation qui n'est pas UP-résiliente présentée dans l'exemple 20, aucune propagation ne peut être faite.

7.2.4 UP-résilience des motifs existants

Des résultats empiriques [LMP07] ont montré par le passé l'efficacité pratique des motifs 2.15, 2.16 et 2.17 dans le schéma d'apprentissage des solveurs de type séparation et évaluation, mais jusqu'à présent il n'y avait pas de résultat théorique pour l'expliquer. La propriété suivante montre que les transformations des parties des sous-ensembles inconsistants correspondant à ces motifs sont UP-résilientes.

Propriété 11. Soit Φ une formule et ψ un sous-ensemble inconsistant de Φ. Pour tout $\psi' \subset \psi$ tel que ψ' correspond à un des motifs 2.15, 2.16 ou 2.17 il existe une séquence S des variables de ψ telle que la transformation $\Theta(\psi, S)$ est UP-résiliente pour les littéraux de ψ'.

Démonstration. Considérons la partie ψ' d'un sous-ensemble inconsistant ψ correspondant au motif 2.15 :

$$\psi' = \{c_1 = \{l_1, l_2\},\ c_2 = \{l_1, \overline{l_2}\}\}$$

On suppose sans perte de généralité que le littéral l_1 a été propagé par une clause c_3. Il y a deux séquences de propagations possibles, menant à des affectations différentes des variables $var(l_1)$ et $var(l_2)$: $\langle \ldots, l_1, l_2 \rangle$ et $\langle \ldots, l_1, \overline{l_2} \rangle$. Les graphes d'implications représentant ces séquences de propagations sont présentés dans la figure 7.4.

FIGURE 7.4 – Graphes d'implications décrivant les étapes de propagations présentées dans la démonstration de la propriété 11.

Les voisinages possibles des littéraux apparaissant dans ces graphes d'implications sont les suivants (on ignore les prédécesseurs de l_1 qui ne sont pas utiles à cette démonstration) :

$$pneigh_{\psi'}(l_1) = \{\{l_2, \square\}, \{\overline{l_2}, \square\}\}$$
$$pneigh_{\psi'}(l_2) = \{\{l_1, \square\}\}$$
$$pneigh_{\psi'}(\overline{l_2}) = \{\{l_1, \square\}\}$$

Tous ces voisinages incluent la clause vide $\square$, par conséquent la transformation par max-résolution de ψ' est bien UP-résiliente.

Le même raisonnement peut être réalisé pour les motifs 2.16 et 2.17, en vérifiant pour chaque littéral l_i apparaissant dans les clauses du motif que pour tout $n_k \in pneigh_\psi(l_i)$ on a $\square \in n_k$ ou l_i peut être propagé dans $\Theta(\psi, S)|_{n_k}$. □

7.3 Étude expérimentale

Nous avons implémenté dans AHMAXSAT une procédure simple permettant de calculer le pourcentage d'UP-résilience des transformations. Nous générons l'ensemble des voisinages de chaque littéral l_i apparaissant dans les sous-ensembles inconsistants puis nous vérifions si l_i est propagé dans la formule transformée lorsque ses voisinages sont satisfaits. Cette implémentation naïve est coûteuse en temps de calcul et résulte en une augmentation d'environ 25% du temps de résolution du solveur. Son objectif est d'évaluer les mécanismes de transformation et d'apprentissage des sous-ensembles inconsistants, pas d'être compétitive.

Les variantes d'AHMAXSAT présentées ci-dessous sont testées sur le benchmark et selon le protocole expérimental présentés dans la section 2.6.

7.3.1 Impact de l'ordre d'application des étapes de max-résolution sur l'UP-résilience

Nous évaluons tout d'abord l'impact de l'ordre d'application de étapes de max-résolution sur l'UP-résilience des transformations. Nous comparons deux variantes :

— AHMAXSAT^{RPO} utilise l'heuristique RPO et applique les étapes de max-résolution dans l'ordre inverse des propagations,

— AHMAXSAT^{SIR} utilise l'heuristique SIR, qui privilégie les étapes de max-résolution produisant les plus petits résolvant intermédiaires.

Le fonctionnement de ces deux variantes est détaillé dans le chapitre 4 de cette thèse.

Les résultats sont présentés dans la table 7.1. On peut observer que le pourcentage moyen d'UP-résilience des transformations est sensiblement plus élevé avec l'heuristique SIR qu'avec l'heuristique RPO (64,8% contre 57,1% sur l'ensemble du benchmark). Par conséquent, moins de nœuds de l'arbre de recherche sont explorés (en moyenne 80036 contre 93175), le temps moyen de résolution est plus faible (97,02 secondes contre 128,36) et plus d'instances sont résolues (1395 contre 1364). Cela montre que l'ordre d'application des étapes de max-résolution a un impact important sur l'UP-résilience des transformations et donc sur les performances des solveurs.

7.3.2 Impact du schéma d'apprentissage sur l'UP-résilience

Dans la seconde série de tests, nous évaluons l'impact des schémas d'apprentissage sur l'UP-résilience des transformations apprises. Nous considérons les variantes suivantes :

— AHMAXSAT^{IRS} utilise le schéma d'apprentissage de MINIMAXSAT [HLO08], que nous désignons par IRS (pour *intermediary resolvents size*). Les transformations par max-résolution sont conservées dans la sous-partie de l'arbre de recherche si et seulement si toutes les clauses résolvantes intermédiaires obtenues pendant la transformation sont de taille inférieure à quatre.

— AHMAXSAT^{PAT} utilise un schéma d'apprentissage basé sur les motifs 2.15, 2.16 et 2.17, comme les solveurs WMAXSATZ [LMP07] et AKMAXSAT [Küg10]. Nous désignons ce schéma par PAT (pour *patterns*).

— AHMAXSAT^{PAT+} utilise également un schéma d'apprentissage basé sur les motifs, mais il inclut les ensembles de motifs $\{3^t, 4^t, 5^t\}$-UCS introduits dans le chapitre 6 en plus des motifs classiques 2.15, 2.16 et 2.17. Nous désignons ce schéma par PAT+.

— AHMAXSAT^{UPR} utilise un schéma d'apprentissage basé sur l'UP-résilience. Seules les transformations UP-résilientes sont apprises dans la sous-partie de l'arbre de recherche. Nous déignons ce schéma par UPR.

La table 7.2 montre les résultats obtenus. On peut tout d'abord observer que le schéma d'apprentissage IRS a un comportement très différent de ceux de PAT et PAT+. Il apprend en moyenne 73,9% des transformations tandis que PAT et PAT+ apprennent moins de 25% d'entre elles (colonnes L). Le pourcentage moyen d'UP-résilience des transformations apprises est relativement bas (79%, non représenté dans la table 7.2) en comparaison de ceux des schémas PAT et PAT+ (respectivement 100% et 98%). Par conséquent, la propagation unitaire détecte moins efficacement les sous-ensembles inconsistants avec le schéma IRS qu'avec les schémas PAT et PAT+. Il explore donc plus de nœuds de l'arbre de recherche (en moyenne respectivement +244% et +351% par rapport à PAT et PAT+, colonnes D), le nombre d'instances résolues est plus faible et le temps moyen de résolution plus élevé (colonnes S(T)). Ces résultats permettent de conclure que le schéma d'apprentissage IRS, tel qu'implémenté dans la variante AHMAXSAT^{IRS}, ne permet pas de contrôler efficacement l'impact de l'apprentissage sur le mécanisme de la propagation unitaire.

La comparaison des schémas d'apprentissage PAT et PAT+ montre que ce dernier à un pourcentage de transformations apprises légèrement supérieur (24,5% contre 21,5%,

TABLE 7.1 – Comparaison de l'impact de l'ordre d'application des étapes de max-résolution sur l'UP-résilience des transformations. Les deux première colonnes montrent les classes d'instances et le nombre d'instances par classe. Pour chaque variante, les colonnes S(T), D et UPR donnent respectivement le nombre d'instances résolues avec entre parenthèse le temps moyen de résolution, le nombre moyens de nœuds explorés et le pourcentage moyen d'UP-résilience des transformations. Les colonnes marquées avec une étoile ne prennent en compte que les instances résolues par toutes les variantes.

Classes d'instances			#	AHMAXSAT^{SIR}			AHMAXSAT^{RPO}		
				S(T*)	D*	UPR*	S(T*)	D*	UPR*
ms	crafted	bipartite	100	100(117,58)	34915	45,2%	100(136,44)	40608	41,6%
		maxcut	67	56(55,82)	171380	79,4%	56(64,55)	208687	73,1%
		set-covering	10	0(-)	-	-	0(-)	-	-
	random	highgirth	82	6(1017,82)	3745740	68,8%	2(1349,10)	4198100	57,9%
		max2sat	100	100(115,48)	45444	54,4%	100(157,59)	60277	48,3%
		max3sat	100	100(290,07)	328030	64,2%	99(347,74)	408764	54,5%
		min2sat	96	96(3,27)	1046	55,3%	96(3,93)	1188	51,6%
pms	crafted	frb	25	5(181,58)	585949	84,5%	5(184,82)	609626	79,7%
		job-shop	3	0(-)	-	-	0(-)	-	-
		maxclique	158	132(29,21)	19493	85,9%	133(26,15)	18686	83,0%
		maxone	140	107(38,19)	133747	86,1%	107(43,45)	131269	75,3%
		min-enc/kbtree	42	30(191,23)	57989	70,7%	20(661,99)	164731	58,8%
		pseudo/miplib	4	2(0,03)	288	83,8%	2(0,03)	271	77,9%
		reversi	44	7(2,12)	56	59,8%	5(6,78)	69	45,8%
		scheduling	5	0(-)	-	-	0(-)	-	-
	random	min2sat	60	55(115,72)	6200	40,9%	50(171,87)	9245	33,2%
		min3sat	60	58(324,52)	85114	52,7%	57(407,59)	107054	44,1%
		pmax2sat	60	60(4,42)	1021	61,4%	60(6,09)	1349	53,8%
		pmax3sat/hi	30	30(83,47)	84944	58,9%	30(100,55)	99455	47,7%
wpms	crafted	auction	40	40(169,35)	310021	92,4%	40(204,42)	300832	80,2%
		CSG	10	4(589,24)	54813	76,1%	4(547,29)	54569	50,1%
		frb	34	14(101,81)	212369	85,2%	14(74,36)	220618	81,3%
		min-enc	74	53(54,73)	18863	68,1%	50(151,67)	8055	51,3%
		pseudo/miplib	12	2(0,08)	52	70,6%	2(1,16)	52	37,7%
		ramsey	15	4(69,42)	157559	82,9%	4(74,05)	160686	76,9%
		random-net	32	3(499,20)	391871	92,1%	2(915,15)	823476	77,0%
		set-covering	45	10(-)	-	-	0(-)	-	-
		wmaxcut	48	42(56,47)	29387	79,9%	46(60,52)	34640	73,0%
	random	wmax2sat	120	120(61,64)	3993	53,5%	120(80,77)	4794	47,5%
		wmax3sat	40	40(142,42)	36836	61,9%	40(175,07)	46786	50,4%
		wpmax2sat	90	89(7,31)	579	59,2%	90(10,59)	643	51,6%
		wpmax3sat/hi	30	30(123,23)	30451	54,3%	30(162,61)	37451	40,0%
Ensemble			1776	1395(97,02)	80036	64,8%	1364(128,36)	93175	57,1%

TABLE 7.2 – Comparaison de l'impact des schémas d'apprentissage IRS, PAT, PAT+ et UPR. Les deux première colonnes montrent les classes d'instances et leur nombre d'instances. Pour chaque variante, les colonnes S(T), D et L donnent respectivement le nombre d'instances résolues avec entre parenthèse le temps moyen de résolution, le nombre moyen de nœuds explorés et le pourcentage moyen de transformations apprises. Les colonnes marquées avec une étoile ne considèrent que les instances résolues par toutes les variantes.

Classes d'instances		#	AHMAXSAT^{IRS}			AHMAXSAT^{PAT}			AHMAXSAT^{PAT+}			AHMAXSAT^{UPR}		
			S(T*)	D*	L*	S(T*)	D*	L*	S(T*)	D*	L*	S(T*)	D*	L*
ms	crafted	bipartite	100	1(1678,2)	2476529	76,3%	100(25,82)	7111	6,3%	100(24,32)	7159	6,2%	100(23,47)	6873	6,3%
		maxcut	67	55(100,14)	542060	90,3%	56(78,56)	244495	28,5%	56(50,07)	173361	36,3%	56(45,57)	162782	41,5%
		set-covering	10	0(-)	-	-	0(-)	-	-	0(-)	-	-	0(-)	-	-
	random	highgirth	82	3(1376,89)	5821506	72,3%	6(1154,65)	4015493	27,1%	6(1146,41)	4116346	28,2%	6(1140,21)	4120315	31,1%
		max2sat	100	16(732,36)	1384544	76,7%	100(8,37)	3321	11,9%	100(7,9)	3363	13,1%	100(7,73)	3150	12,3%
		max3sat	100	66(543,87)	1928894	85,9%	98(139,06)	147000	13,2%	100(111,21)	134846	17,5%	100(114,31)	134644	16,9%
		min2sat	96	87(109,92)	191822	76,1%	96(1,21)	414	17,0%	96(1,13)	410	17,0%	96(1,2)	415	16,9%
pms	crafted	frb	25	5(165,63)	639395	86,4%	5(430,65)	1364465	39,9%	5(181,58)	585949	49,4%	5(156,59)	521263	52,3%
		job-shop	3	0(-)	-	-	0(-)	-	-	0(-)	-	-	0(-)	-	-
		maxclique	158	132(24,58)	43769	94,4%	133(25,93)	24506	61,5%	132(28,69)	19382	66,9%	134(17,97)	17486	67,3%
		maxone	140	107(26,45)	157230	73,3%	107(38,6)	148070	12,6%	107(38,19)	133747	12,8%	107(23,75)	120250	27,3%
		min-enc/kbtree	42	15(529,73)	533014	53,2%	30(128,35)	49425	3,8%	30(142,41)	52920	4,0%	32(109,88)	46916	7,3%
		pseudo/miplib	4	2(0,08)	2528	94,9%	2(0,03)	268	25,4%	2(0,03)	288	31,1%	2(0,04)	465	39,5%
		reversi	44	7(93,3)	4619	29,2%	7(95,73)	4210	12,4%	7(99,57)	4067	14,3%	7(87,02)	4053	17,3%
		scheduling	5	0(-)	-	-	0(-)	-	-	0(-)	-	-	0(-)	-	-
	random	min2sat	60	24(441,42)	112557	63,7%	56(13,06)	1241	5,4%	55(14,94)	1241	5,4%	56(13,2)	1247	5,4%
		min3sat	60	21(670,79)	710507	72,7%	58(51,87)	16959	13,6%	58(52,07)	15453	13,6%	58(47,97)	15455	14,0%
		pmax2sat	60	59(126,18)	101463	69,7%	60(4,16)	987	18,5%	60(4,13)	962	20,4%	60(4,15)	975	19,7%
		pmax3sat/hi	30	29(291,09)	958427	79,3%	30(72,14)	74046	11,6%	30(67,64)	69971	13,9%	30(68,68)	72809	13,8%
wpms	crafted	auction	40	39(216,7)	296973	82,4%	40(145,56)	478389	35,2%	40(166,79)	316119	42,1%	40(142,71)	271838	58,9%
		CSG	10	1(710,27)	85274	99,7%	4(773,04)	84860	25,4%	4(859,04)	84860	25,4%	4(578,74)	82248	32,4%
		frb	34	14(55,59)	231672	89,7%	14(145,88)	492276	44,3%	14(101,81)	212369	53,0%	14(61,69)	188998	55,8%
		min-enc	74	51(136,27)	20317	29,7%	54(99,2)	19591	6,1%	54(109,78)	18615	6,2%	54(89,21)	63529	13,8%
		pseudo/miplib	12	1(4,97)	43	-	3(1,47)	37	-	3(1,47)	37	-	3(1,26)	35	-
		ramsey	15	4(145,77)	164184	81,3%	4(65,92)	158529	53,2%	4(69,42)	157559	54,6%	4(65,62)	162982	56,9%
		random-net	32	2(627,37)	1018377	74,2%	1(1686,64)	1253990	3,4%	3(374,2)	335161	9,1%	3(656,23)	1354993	25,1%
		set-covering	45	10(976,49)	4695	9,7%	10(954,54)	3655	4,2%	10(1013,31)	3655	4,2%	10(936,04)	3655	4,2%
		wmaxcut	48	42(52,4)	42579	88,5%	46(103,42)	58039	23,2%	46(56,47)	29387	36,7%	46(44,16)	25809	42,3%
	random	wmax2sat	120	81(492,51)	97322	63,0%	120(24,82)	1902	9,0%	120(23,89)	1736	10,5%	120(23,61)	1816	9,4%
		wmax3sat	40	39(253,31)	151542	81,7%	40(146,39)	38306	7,7%	40(131,51)	34237	11,7%	40(128,51)	34635	12,0%
		wpmax2sat	90	90(63,35)	4023	60,0%	90(7,17)	575	16,7%	90(7,23)	575	18,1%	90(6,7)	577	17,8%
		wpmax3sat/hi	30	30(237,83)	110777	64,8%	30(125,09)	31591	5,2%	30(123,23)	30451	6,4%	30(121,34)	30906	7,2%
Ensemble		1776	1033(210,69)	327864	73,9%	1400(72,51)	95222	21,5%	1402(65,94)	72683	24,5%	1407(58,27)	71279	27,6%

colonnes L) tout en conservant un pourcentage moyen d'UP-résilience des transformations apprises élevé (98% contre 100%). Par conséquent, le nombre moyen de nœuds explorés ainsi que le temps moyen de résolution sont légèrement améliorés.

Il est aussi intéressant d'observer que le pourcentage moyen de transformations apprises avec le schéma PAT+ (24,5%) est sensiblement plus faible que celui du schéma UPR (27,6%). Cela montre que les motifs utilisés actuellement ne permettent pas de détecter toutes les transformations UP-résiliente et donc que les schémas d'apprentissage peuvent être améliorés.

7.3.3 Schéma d'apprentissage basé sur l'UP-résilience

Nous avons implémenté un nouveau schéma d'apprentissage basé sur le pourcentage minimum d'UP-résilience autorisé dans les transformations apprises des sous-ensembles inconsistants (notée $\%UPR_{IS}$) et des UCS (notée $\%UPR_{UCS}$). Nous avons testé ce nouveau schéma d'apprentissage en faisant varier les valeurs de $\%UPR_{IS}$ et $\%UPR_{UCS}$ respectivement de 0 à 100 et de 40 à 100 par pas de 10. Le benchmark utilisé pour cette série d'expérimentation est composé des instances non-valuées et valuées, aléatoires et *crafted* du benchmark de la Max-SAT Evaluation 2013 [1]. Refaire ces tests sur le benchmark présenté en section 2.6 aurait nécessité 136906 lancements uniques de notre solveur et nous n'avons pas jugé utile de les réaliser. Cela n'affecte pas la porté des résultats obtenus.

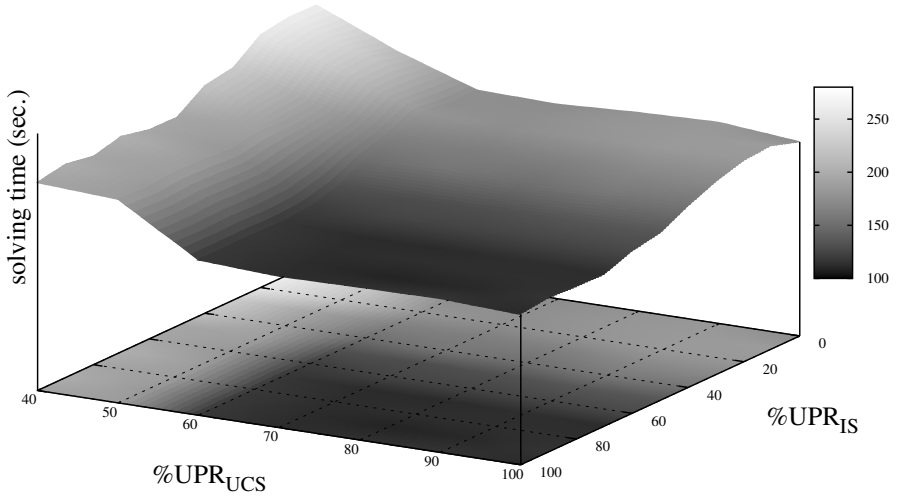


FIGURE 7.5 – Impact du pourcentage minimum d'UP-résilience autorisé dans l'apprentissage des transformations des sous-ensembles inconsistants et des UCS sur le temps moyen de résolution.

Les résultats sont présentés dans la figure 7.5 et 7.6. On peut observer que les meilleurs résultats, que se soit en termes de temps moyen de résolution ou en nombre moyen de nœuds explorés, sont obtenus avec des valeurs de $\%UPR_{IS}$ et $\%UPR_{UCS}$ allant de

1. Disponibles à l'adresse `http://www.maxsat.udl.cat/13`.

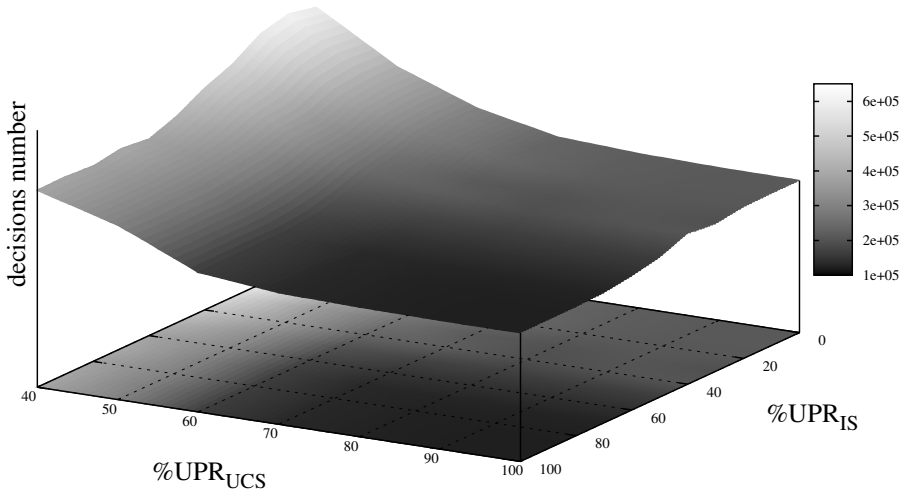


FIGURE 7.6 – Impact du pourcentage minimum d'UP-résilience autorisé dans l'apprentissage sur le nombre de nœuds explorés.

60% à 100%. Dans ces intervalles de valeurs, le pourcentage moyen d'UP-résilience des transformations apprises est toujours supérieur à 90%. Lorsque ces valeurs sont inférieures à 60%, le nombre moyen de nœuds explorés et le temps moyen de résolution croissent significativement. Ces résultats montrent que l'UP-résilience permet de quantifier de manière précise l'impact des transformations sur l'efficacité de la propagation unitaire.

7.4 Conclusions

Nous avons introduit dans ce chapitre la notion d'UP-résilience des transformations, qui permet de quantifier l'impact de la max-résolution sur le mécanisme de la propagation unitaire. Nous avons montré que, selon le critère de l'UP-résilience, les motifs utilisés dans les schémas d'apprentissage des solveurs de type séparation et évaluation existant n'affectent pas l'efficacité de la propagation unitaire. Ceci contribue à expliquer d'un point de vue théorique l'efficacité des ces schémas d'apprentissage qui avait été démontré uniquement de manière empirique jusqu'alors. Les résultats de l'étude expérimentale que nous avons menée montrent la pertinence de la notion d'UP-résilience. Ils montrent également que :

— l'ordre d'application des étapes de max-résolution lors de la transformation des sous-ensembles inconsistants a un impact sur l'UP-résilience des transformations et que
— les schémas d'apprentissage existant ne permettent pas d'apprendre toutes les transformations UP-résiliente.

Ces résultats ouvrent de nouvelles perspectives. Parmi elles, il serait intéressant de développer de nouveaux ordres d'application de la max-résolution visant à accroître le pourcentage d'UP-résilience des transformations. Les schémas d'apprentissage existants peuvent également être améliorés, soit en établissant une caractérisation des transforma-

tions UP-résilientes qui puisse être vérifiée efficacement ou en complétant les ensembles de motifs existants. Maintenant que les effets des transformations par max-résolution sont mieux compris, on peut envisager d'appliquer l'apprentissage non seulement vers le bas de l'arbre de recherche, comme c'est actuellement le cas, mais également vers le haut. Les bénéfices escomptés seraient de réduire encore davantage la redondance dans le calcul de la borne inférieure et de permettre d'exploiter certaines propriétés structurelles des instances.

Chapitre 8

Description d'AHMAXSAT et comparaison expérimentale

Nous faisons dans ce chapitre une présentation du solveur de type séparation et évaluation AHMAXSAT développé pendant cette thèse et qui nous a servi de plateforme pour évaluer empiriquement les travaux présentés dans les chapitres 3 à 7. Nous donnons une vue d'ensemble de son fonctionnement et discutons des interactions entre les composants présentés précédemment. Puis, nous présentons les résultats de l'étude expérimentale que nous avons conduite. Nous y comparons AHMAXSAT aux solveurs de type séparation et évaluation de l'état de l'art et, plus généralement, à quelques uns des solveurs complets les plus performants.

8.1 Vue d'ensemble de notre solveur AHMAXSAT

Nous avons développé dans le cadre de cette thèse un nouveau solveur séparation et évaluation, que nous avons nommé AHMAXSAT. Certains des travaux réalisés dans le cadre de cette thèse (cf. chapitres 3 et 5) nécessitant des aménagements profonds dans l'architecture et les structures de données des solveurs séparation et évaluation, leur intégration dans un solveur existant aurait nécessité un long temps de développement. Il nous a paru plus pertinent d'implémenter un nouveau solveur en prenant en considération ces nouveaux éléments dès les premières étapes de sa construction.

L'architecture générale d'AHMAXSAT est similaire à celle présentée dans l'algorithme 2.1. Il explore l'ensemble de l'arbre de recherche. À chaque nœud, il calcule la borne inférieure et compare la valeur obtenue à celle de la borne supérieure pour déterminer s'il est utile d'examiner la branche courante de l'arbre de recherche. Nous décrivons dans la suite de cette section les principaux éléments d'AHMAXSAT : les règles d'inférence sémantiques qu'il utilise pour étendre l'interprétation courante, son heuristique de branchement et sa fonction de calcul de la borne inférieure. Nous discuterons également de sa gestion des clauses dures de la formule et des interactions entre les différents composants présentés dans les chapitres précédents.

8.1.1 Règles d'inférence sémantiques

AHMAXSAT utilise à chaque nœud de l'arbre de recherche les règles d'inférence sémantiques présentées dans les sections 2.2 et 2.3.3 : la propagation unitaire réelle (HUP), la règle des littéraux purs (PL) et la la règle des clauses unitaires dominantes (DUC). Contrairement aux solveurs de type séparation et évaluation existants qui appliquent ces règles seulement une fois avant le calcul de la borne inférieure, AHMAXSAT applique dynamiquement HUP, PL et DUC durant le calcul de la LB. Ceci est rendu possible par la max-résolution locale (cf. chapitre 5), qui permet de conserver l'équivalence de la formule pendant toute la durée du calcul de la borne inférieure.

Cela peut permettre de fixer la valeur de certaines variables plus haut dans l'arbre de recherche que ne le feraient les autres solveurs séparation et évaluation. Ces affectations réduisent des clauses de la formule qui peuvent être utilisées par la propagation unitaire pour détecter davantage de sous-ensembles inconsistants. Aussi, cela peut conduire à une amélioration de la qualité de la borne inférieure.

Le schéma d'application de la propagation unitaire MPS (cf. chapitre 3) est également utilisé par AHMAXSAT pour appliquer la propagation unitaire réelle. Il permet de prendre en compte la somme des poids des sources de propagation plutôt que le poids de la première source. Par exemple, si un littéral l_i est tel que $LB + \sum_{c_j \in src(l_i)} poids(c_j) \geq UB$, alors l_i peut être satisfait par application de HUP.

8.1.2 Heuristique de branchement

L'heuristique de branchement d'AHMAXSAT est proche de celle de WMAXSATZ. Pour une formule Φ et une interprétation courante I, il choisit la variable non affectée qui maximise :

$$score(x_i) = lscore(x_i) * lscore(\overline{x_i}) + lscore(x_i) + lscore(\overline{x_i})$$

avec :

$$lscore(l_i) = 2 * app_1(l_i) + 4 * app_2(l_i) + app_3(l)$$

où $app_k(l_i)$ est la somme des poids des clauses de taille k de la formule $\Phi|_I$ dans lesquelles l_i apparaît. La valeur donnée à la variable x_i choisie dépend de la valeur de $lscore()$ de ses littéraux. Si $lscore(x_i) \geq lscore(\overline{x_i})$ alors x_i est fixée à $faux$ et sinon elle est fixée à $vrai$.

8.1.3 Calcul de la borne inférieure

La fonction de calcul de la borne inférieure utilisée par AHMAXSAT est présentée dans l'algorithme 8.1. Il commence par essayer d'étendre l'interprétation courante I avec les règles d'inférence sémantiques HUP, PL et DUC (ligne 4). Puis, itérativement, il applique SUP* avec le schéma MPS et applique un traitement aux sous-ensembles inconsistants détectés (lignes 6-9). Lorsque SUP ne détecte plus aucun conflit, l'algorithme applique la technique des littéraux contradictoires. Il sélectionne une variable non affectée dans la liste des variables candidates (ligne 11-12). Puis il détecte les sous-ensembles inconsistants par SUP* et les transforme jusqu'à saturation. Si des sous-ensembles inconsistants ont été détectés dans $\Phi|_{I\cup\{l\}}$, alors leur transformation a eu pour effet d'ajouter des clauses résolvantes qui sont unitaires sous l'interprétation I. Dans ce cas, l'algorithme retourne à l'application de SUP* (ligne 6). Sinon, il choisit une nouvelle variable non affectée et continue l'application de FL (ligne 11). Si l'algorithme a appliqué FL à toutes les variables candidates sans parvenir à générer de nouvelles clauses unitaires, alors il retourne à la ligne 4 et vérifie si I peut être étendu par application des règles d'inférence sémantiques. Ce processus est répété jusqu'à ce qu'aucun sous-ensemble inconsistant ne

Algorithme 8.1 : AHMAXSAT : calcul de la borne inférieure

Data : Une formule CNF Φ définie sur un ensemble de variables booléennes X et une interprétation partielle I.
Result : La formule transformée Φ et un entier positif LB.

```
1  begin
2      candidats_FL ← X \ I;
3      repeat
           // Règles d'inférence pour étendre l'interprétation courante
4          I ← règles_inférence_HUP_PL_DUC(Φ, I, X \ I);
5          repeat
               // Détection des sous-ensembles inconsistant par SUP*
6              G = (V, A) ← SUP*(Φ|_I);
7              while □ ∈ V do
8                  Φ ← traitement_conflit(Φ, G);
9                  G = (V, A) ← SUP*(Φ|_I);
               // Ici SUP est saturée, on applique FL
10             repeat
11                 x ← candidats_FL;
12                 l ← littéral de x apparaissant le plus dans des clauses binaires;
13                 G = (V, A) ← SUP*(Φ|_{I∪{l}});
14                 while □ ∈ V do
15                     Φ ← traitement_conflit(Φ, G));
16                     G = (V, A) ← SUP*(Φ|_{I∪{l}});
17             until (Φ|_I contient des clauses unitaires) ou (candidats_FL = ∅);
18         until Φ|_I ne contienne plus de clause unitaire;
19     until Φ|_I = Φ'|_I;
20     LB ← Σ_{c_j=□∈Φ|_I} poids(c_j);
21     Φ ← restauration_changements_locaux(Φ);
22     return (Φ, LB);
```

puisse être détecté par SUP*, qu'il ne reste plus aucun candidat à l'application de FL et que l'interprétation I ne puisse plus être étendue. L'algorithme calcule alors la borne inférieure en comptant la somme des poids des clauses de la formule falsifiées par l'interprétation courante I (ligne 20). Enfin, il défait les modifications locales apportées à la formule (ligne 21).

Les fonctions suivantes sont utilisées dans l'algorithme. `règles_inférence_HUP_PL_DUC()` vérifie pour chaque variable non affectée si une règle d'inférence sémantique (HUP, PL ou DUC) peut être appliquée. Si c'est le cas, elle met à jour l'interprétation I. `SUP*()` applique la propagation unitaire simulée avec deux queues séparées comme décrit dans la section 2.3.4.1 et avec le schéma de propagation MPS présenté dans le chapitre 3. Quand elle détecte un conflit ou que la formule ne contient plus aucune clause unitaire, elle renvoie le graphe d'implications plein décrivant les étapes de propagations unitaires simulées réalisées. La fonction `traitement_conflit()` est décrite ci-dessous. `restauration_changements_locaux()` défait simplement les changements locaux de la formule réalisés pendant le calcul de la borne inférieure, puis renvoie la formule modifiée.

La fonction `traitement_conflit()` (Algorithme 8.2) prend en entrée deux paramètres : la formule Φ et un graphe d'implications plein conflictuel. Elle commence par convertir le graphe d'implications plein G en un graphe d'implications classique G' en choisissant, pour chaque littéral propagé participant au conflit, la source de propagation selon l'heuristique SIS décrite dans le chapitre 3 (ligne 2). Puis, elle calcule la séquence S d'étapes de max-résolution selon l'heuristique SIR présentée dans le chapitre 4 (ligne 3). Elle analyse le graphe d'implications G' et sépare le sous-ensemble inconsistant en deux sous-parties : ψ_{st} qui correspond au motif 2.17 ou à un des $\{2, 3, 4^t, 5^t\}$-UCS (cf. sections 2.3.4.3 et 6.2)et l'autre partie ψ_{tmp}. Enfin, elle transforme les clauses du sous-ensemble inconsistant par max-résolution selon la séquence S. Les transformations sont mémorisées dans deux queues séparées, une pour les changements locaux (qui seront restaurés à la fin du calcul de la borne inférieur) et une pour les changements conservés dans le sous-arbre (qui seront restaurés lors des retours-arrières). La fonction renvoie la formule transformée.

Algorithme 8.2 : AHMAXSAT : analyse et traitement des conflits.

Data : Une formule CNF Φ et un graphe d'implications plein conflictuel $G = (V, A)$.
Result : La formule transformée Φ.

```
1 begin
2     G' = (V', A') ← conversion_graphe_implications_plein(G);
3     S ← ordre_application_max_resolution(G');
4     (ψ_st, ψ_tmp) ← detection_motifs(G');
5     Φ ← application_max_resolution(Φ,G',S,ψ_st,ψ_tmp);
6     return Φ;
```

8.1.4 Gestion des clauses dures

Notre solveur n'utilise aucune technique particulière sur la partie dure des instances Max-SAT partiel, à l'exception de la règle de la propagation unitaire réelle. Son comportement est donc similaire à celui d'un algorithme DPLL pour SAT.

Nous lui ajouterons dans un futur proche des techniques de résolution utilisées dans les solveurs SAT modernes, comme l'apprentissage de clauses, les retours arrières non-chronologiques ou les heuristiques de branchement basées sur les apparitions des variables dans les conflits.

8.1.5 Interactions entre les composants d'AHMAXSAT

Nous avons présenté dans les chapitres 3 à 7 les quatre principaux composants originaux d'AHMAXSAT. Nous discutons maintenant de leurs interactions. Nous rappelons brièvement le fonctionnement de chacun d'entre eux et décrivons l'effet qu'ont sur eux les autres composants.

Schéma de propagation MPS et heuristique SIS Le schéma de propagation MPS, qui considère toutes les sources de propagation des variables, a pour effet de réduire la redondance dans l'application de la propagation unitaire (SUP et FL). Par conséquent, plus le solveur fait d'étapes de propagation unitaire et plus l'impact du schéma MPS sur les performances du solveur sera important.

La max-résolution locale, comme nous l'avons vu dans le chapitre 5, a pour effet d'augmenter le nombre d'étapes de propagation unitaire réalisées à chaque nœud de l'arbre de recherche. Par conséquent, l'impact du schéma MPS s'en trouve probablement accrut. À l'inverse, l'augmentation de l'apprentissage entraîné par les motifs UCS réduit légèrement la redondance dans la détection des sous-ensembles inconsistants par propagation unitaire. Cela a pour conséquence de réduire (légèrement) l'impact du schéma MPS.

Heuristique SIR Le principal effet de l'heuristique SIR est de réduire le nombre et la taille des clauses de compensation produites par la max-résolution. Il y a par conséquent une forte corrélation entre l'impact de l'heuristique SIR sur les performances du solveur et la fréquence d'application de la règle de la max-résolution.

Cette heuristique montre son plein potentiel lorsqu'elle est utilisée en combinaison avec la max-résolution locale. Sans max-résolution locale, les sous-ensembles inconsistants ne sont transformés par max-résolution que lorsqu'ils remplissent les critères d'apprentissage, c'est-à-dire dans environ 20% des cas. L'impact de l'heuristique SIR sur le comportement du solveur est alors marginal.

Max-résolution locale Le principal inconvénient de la max-résolution locale repose sur le traitement, coûteux en temps de calcul, qui est appliqué sur les sous-ensembles inconsistants. Toute méthode permettant de réduire le coût de ce traitement améliore l'impact de la max-résolution locale sur les performances du solveur.

C'est notamment le cas de l'heuristique SIR, qui permet de réduire le nombre et la taille des clauses de compensation ajoutées à la formule. Le schéma d'apprentissage basé sur les UCS rend la détection et le traitement des sous-ensembles inconsistants plus incrémental, et donc limite lui aussi l'impact des inconvénients de la max-résolution locale.

Schéma d'apprentissage étendu La fréquence de détection des motifs UCS dans les sous-ensembles inconsistants dépend de la taille des clauses de la formule. Plus la

formule contient de clauses de petites tailles, plus la probabilité d'apparition des motifs $\{2, 3, 4^t, 5^t\}$-UCS augmente.

L'heuristique SIR, qui permet de réduire significativement le nombre et la taille des clauses de compensation, peut donc permettre d'accroître légèrement le nombre d'UCS détectés, et ainsi de réaliser davantage d'apprentissage. Comme nous l'avons vu dans le chapitre 5, la max-résolution locale a pour effet de réduire le pourcentage d'apprentissage réalisé par AHMAXSAT. Par conséquent, cela limite l'impact du schéma d'apprentissage étendu sur les performances du solveur.

8.2 Étude expérimentale

Nous comparons dans cette section notre solveur AHMAXSAT avec les solveurs de type séparation et évaluation récents ainsi qu'avec certains des autres solveurs complets les plus performants. Nous incluons dans cette comparaison deux variantes de notre solveur :

— AHMAXSAT-ALL inclut tous les composants présentés dans les chapitres 3 à 7.

— AHMAXSAT-NONE n'en inclut aucun.

Ces deux variantes utilisent le schéma de propagation MPS présenté dans le chapitre 3. Nous considérons dans notre comparaison les solveurs de type séparation et évaluation de l'état de l'art suivants :

— [W]MAXSATZ [LMP06, LMP07, LMMP09, LMMP10] inclut les composants les plus récents présentés dans la section 2.3. Son code source est disponible [1] et il a fait l'objet de nombreuses publications. Ces éléments font de [W]MAXSATZ un solveur de référence parmi les algorithmes de type séparation et évaluation pour Max-SAT.

— AKMAXSAT [Küg10] est proche structurellement de [W]MAXSATZ. À notre connaissance, il se distingue principalement par l'intégration de la propagation unitaire simulée et de la technique des littéraux contradictoires en une seule méthode nommée par l'auteur *generalized unit propagation* [Küg10]. Le code source d'AKMAXSAT est également disponible en ligne [2].

— MINIMAXSAT [HLO07, HLO08] est implémenté à partir du code source du solveur SAT MINISAT [ES03, SE08]. Il se distingue de [W]MAXSATZ et d'AKMAXSAT par plusieurs aspects. Un des principaux est l'utilisation de techniques de résolution issue des solveurs SAT CDCL (retour-arrières non-chronologiques, heuristique de branchement VSIDS, etc.) sur les clauses dures des instances. Pour la partie souple des instances, il utilise un schéma d'apprentissage basé sur la taille des clauses résolvantes intermédiaires plutôt que sur des motifs comme les autres solveurs de type séparation et évaluation. A notre connaissance, seul l'exécutable binaire de MINIMAXSAT est disponible en ligne [3].

Tous ont été régulièrement bien classés dans plusieurs catégories lors des *Max-SAT Evaluation* des cinq dernières années. À notre connaissance, il n'existe pas d'autre solveur séparation et évaluation récent. Enfin, nous incluons également trois solveurs SAT itératifs, MAXHS [DB11, DB13a, DB13b], EVA500 [NB14] et OPEN-WBO [MML14], et deux

1. À l'adresse `http://home.mis.u-picardie.fr/~cli/EnglishPage.html`
2. À l'adresse `https://www.uni-ulm.de/en/in/institute-of-theoretical-computer-science/m/alumni/kuegel.html`
3. À l'adresse `http://logos.ucd.ie/wiki/doku.php?id=federico.heras`

solveurs basés sur la reformulation en programmation linéaire en nombres entiers (ILP), ILP-2013 [AG13] et SCIP-MAXSAT [Ach09]. Tous ces solveurs ont été classés dans les trois premières places d'une des catégories d'instances lors de la *Max-SAT Evaluation 2014*. Dans la suite de cette section, nous comparons tout d'abord l'ensemble des solveurs complets en étudiant les particularités et points forts de chacune des classes de solveurs selon les catégories d'instances. Puis, nous comparons les performances des solveurs de type séparation et évaluation.

8.2.1 Comparaison des solveurs complets

Nous avons testé ces solveurs sur les instances aléatoires et *crafted* du benchmark de la *Max-SAT Evaluation 2014* (détaillées dans la section 2.6. Les résultats sont présentés dans la table 8.1, où le nombre d'instances résolues par chaque solveur est donné avec entre parenthèse le temps moyen de résolution. Pour chaque classe d'instances, le meilleur résultat est présenté en gras et le meilleur résultat parmi les solveurs de type séparation et évaluation est souligné.

On peut remarquer que sur l'ensemble du benchmark, les solveurs de type séparation et évaluation résolvent davantage d'instances que les solveurs SAT itératifs ou ceux basés sur la reformulation. Ils dominent clairement les classes d'instances aléatoires ainsi que celles des instances *crafted* non partielles et non valuées. Les résultats sont moins tranchés sur les instances *crafted* partielles (valuées et non-valuées), où les autres solveurs donnent de meilleurs résultats sur la majorité des classes d'instances. On peut noter cependant que même sur ces classes d'instances, les solveurs de type séparation et évaluation se comporte honorablement et que le nombre d'instances qu'ils résolvent n'est pas très éloigné de celui des autres solveurs. À l'inverse les solveurs SAT itératifs et ceux basés sur la reformulation en ILP sont peu efficaces sur les instances aléatoires. Ils résolvent seulement une poignée d'instances, ce qui explique le large écart en nombre d'instances résolues sur l'ensemble du benchmark.

Les performances en demi-teinte des solveurs de type séparation et évaluation sur les instances *crafted* peuvent s'expliquer par leur manque d'efficacité sur la partie dure des instances. Comme nous l'avons vu dans la section 2.6, sur la plupart des classes d'instances Max-SAT partiel (valué ou non-valué), la partie dure des instances est plus importante (en nombre de clauses) que la partie souple. Pour être efficace sur ces instances, les solveurs doivent être capables de parcourir rapidement les solutions de la partie dure. Pour cela, les solveurs de type séparation et évaluation devraient utiliser des techniques modernes de résolution pour SAT, comme l'apprentissage de clauses ou les retours-arrières non-chronologiques. À notre connaissance, ce n'est pas le cas des solveurs de type séparation et évaluation considérés dans cette étude (à l'exception de MINIMAXSAT).

La figure 8.1 montre les temps de résolution cumulés de chacun des solveurs considérés dans l'étude. On peut voir que notre solveur AHMAXSAT-ALL est le plus performant, suivi par les autres solveurs de type séparation et évaluation.

Sur les instances *crafted* (figure 8.2a), MINIMAXSAT résout quelques instances de plus qu'AHMAXSAT-ALL même si ses temps de résolutions sont généralement plus élevés. Le solveur le plus performant suivant est WMAXSATZ2013 suivi des deux solveurs basés sur la reformulation en ILP, ILP-2013 et SCIP-MAXSAT. Si l'on considère les résultats sur les instances Max-SAT *crafted* uniquement (non-partiel, non-valué, figure 8.4a) on peut

TABLE 8.1 – Performances des solveurs sur le benchmark de la MSE 2014. Pour chaque solveur, le nombre d'instances résolues est indiqué avec entre parenthèse le temps moyen de résolution.

	Instance classes	#	AHMAXSAT-ALL	AHMAXSAT-NONE	WMAXSATZ 2013	ARMAXSAT	WMAXSATZ 2009	MINIMAXSAT	IUT-2013	SCIP-MAXSAT	MAXHS	EVA500A	OPEN-WBO
ms	crafted bipartite	100	**100(90,7)**	99(392,2)	99(295)	100(105,2)	99(270,2)	85(641)	0(-)	0(-)	0(-)	0(-)	0(-)
	crafted maxcut	67	**56(44,7)**	55(46,7)	55(56,9)	55(29,8)	55(97,5)	55(143,4)	30(187,3)	24(164)	6(0,7)	9(119,8)	11(86,5)
	crafted set-covering	10	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	**1(46,3)**	0(-)	0(-)
	random highgirth	82	7(1166)	**7(1152,5)**	0(-)	1(1737,3)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)
	random max2sat	100	**100(89,5)**	100(257)	100(177,9)	100(124,4)	97(304,4)	8(1023,4)	0(-)	0(-)	0(-)	0(-)	0(-)
	random max3sat	100	100(231,2)	100(308,9)	100(251,6)	**100(202,1)**	97(383,9)	56(536,7)	2(1609,7)	5(1077)	0(-)	0(-)	0(-)
	random min2sat	96	**96(2,6)**	96(13,7)	96(9,8)	96(6,7)	77(186,5)	61(338,1)	80(116)	60(297,3)	17(481,4)	0(-)	0(-)
pms	crafted frb	25	5(154,2)	5(383,7)	5(178,2)	5(467)	0(-)	5(265)	11(216,7)	12(295,2)	14(347,4)	**25(141,8)**	21(289,7)
	crafted job-shop	3	0(-)	0(-)	0(-)	0(-)	0(-)	1(85,5)	0(-)	0(-)	**3(32,4)**	3(132,8)	3(32,6)
	crafted maxclique	158	133(30,2)	133(35,8)	133(17,8)	133(41,8)	109(84,3)	131(12,5)	132(70,9)	131(111,6)	**135(54,3)**	99(209,7)	85(121,2)
	crafted maxone	140	109(50,1)	111(64,4)	136(34,3)	95(78,1)	137(128,2)	140(6,9)	138(76,7)	139(37,8)	**140(3,7)**	138(106,5)	139(28,4)
	crafted min-enc/kbtree	42	34(476,2)	14(728,8)	10(363,9)	1(269,9)	8(508,7)	14(556,5)	42(197,8)	**42(69,9)**	2(99,7)	5(214,4)	6(26,4)
	crafted pseudo/miplib	4	2(< 0,1)	2(< 0,1)	3(580,8)	3(320,3)	2(0,1)	2(0,4)	4(22,1)	4(31,1)	**4(3,2)**	4(54,9)	4(10,2)
	crafted reversi	44	8(129,2)	7(138,5)	5(2,8)	7(421,4)	7(145,1)	32(56,7)	13(332)	15(307,1)	31(15,1)	32(23,8)	**33(13,8)**
	crafted scheduling	5	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	0(-)	1(1117,5)	**1(505,6)**
	random min2sat	60	**58(186,8)**	44(518,2)	21(786,1)	44(454,6)	43(401,6)	1(414)	48(329,4)	13(747,1)	5(270)	0(-)	7(600,9)
	random min3sat	60	**58(249,9)**	53(464)	49(596,8)	43(602,1)	45(553,9)	6(1034)	13(898,2)	12(1156,6)	0(-)	0(-)	0(-)
	random pmax2sat	60	**60(3,2)**	60(4,6)	59(3,1)	60(11,4)	60(8,5)	56(210,3)	12(574,4)	8(652,6)	0(-)	0(-)	0(-)
	random pmax3sat/hi	30	30(60,1)	30(81)	**30(42,9)**	30(82,3)	30(71)	29(362,2)	9(1028,1)	12(736,3)	0(-)	0(-)	3(515,6)
wpms	crafted auction	40	40(115,5)	40(99)	40(257,1)	40(265,7)	34(217,6)	40(38,8)	40(0,1)	40(0,5)	**40(< 0,1)**	38(5,7)	20(4,7)
	crafted CSG	10	4(452,3)	4(447,9)	1(404)	3(1055,7)	2(967,2)	8(321,8)	4(133,5)	4(35,4)	**10(5,9)**	7(98,5)	5(3,4)
	crafted frb	34	14(58,7)	14(142,3)	14(65,5)	14(157,8)	9(12,2)	14(143,7)	19(124,1)	21(168,4)	23(228,2)	19(63,6)	**30(172,4)**
	crafted min-enc	74	64(108,8)	58(10)	47(53,7)	41(188,4)	51(102,7)	59(42,1)	73(73,9)	74(23,1)	**74(0,8)**	58(3,8)	56(1,3)
	crafted pseudo/miplib	12	4(158,8)	4(290,8)	**5(322,1)**	2(2)	3(131,8)	3(47,3)	3(128,9)	3(378,7)	3(12,1)	5(330,4)	3(3,7)
	crafted ramsey	15	4(44,8)	**5(344,8)**	4(55,2)	4(50)	4(93,2)	4(75,4)	3(187,5)	3(171,8)	1(0,9)	1(0,7)	1(0,2)
	crafted random-net	32	3(572,5)	0(-)	0(-)	1(1122,8)	0(-)	0(-)	25(90,8)	1(1092,1)	**32(9,4)**	12(89,6)	12(8,9)
	crafted set-covering	45	25(46,4)	25(100,9)	0(-)	0(-)	0(-)	19(476,5)	**35(39,1)**	35(51,5)	35(48,6)	9(274,4)	1(0,3)
	crafted wmaxcut	48	**46(49,4)**	44(61,4)	44(108,3)	45(39,6)	42(114,7)	43(162,7)	25(280,9)	20(251,5)	2(20,5)	1(48,6)	0(-)
	random wmax2sat	120	**120(45,5)**	119(196,9)	120(139,3)	120(50,2)	118(277,4)	21(814,8)	2(819,3)	0(-)	0(-)	0(-)	0(-)
	random wmax3sat	40	40(99,5)	40(91,9)	40(136,7)	**40(60)**	40(177,7)	33(561,2)	1(1699,8)	1(915,8)	0(-)	0(-)	0(-)
	random wpmax2sat	90	**90(5,5)**	90(6,4)	89(7,2)	90(27,7)	90(21,7)	83(255,8)	42(171,2)	43(294,7)	4(25,9)	0(-)	0(-)
	random wpmax3sat/hi	30	30(84,1)	30(72,1)	**30(52,8)**	30(78,4)	30(79,6)	29(451,4)	11(653,4)	13(415,2)	0(-)	1(216,6)	0(-)
	Overall	1776	**1440(96,9)**	1389(164,7)	1335(136,3)	1303(119,6)	1289(197,3)	1038(249,4)	817(164,5)	735(176,7)	582(52,6)	467(110,2)	441(76,6)

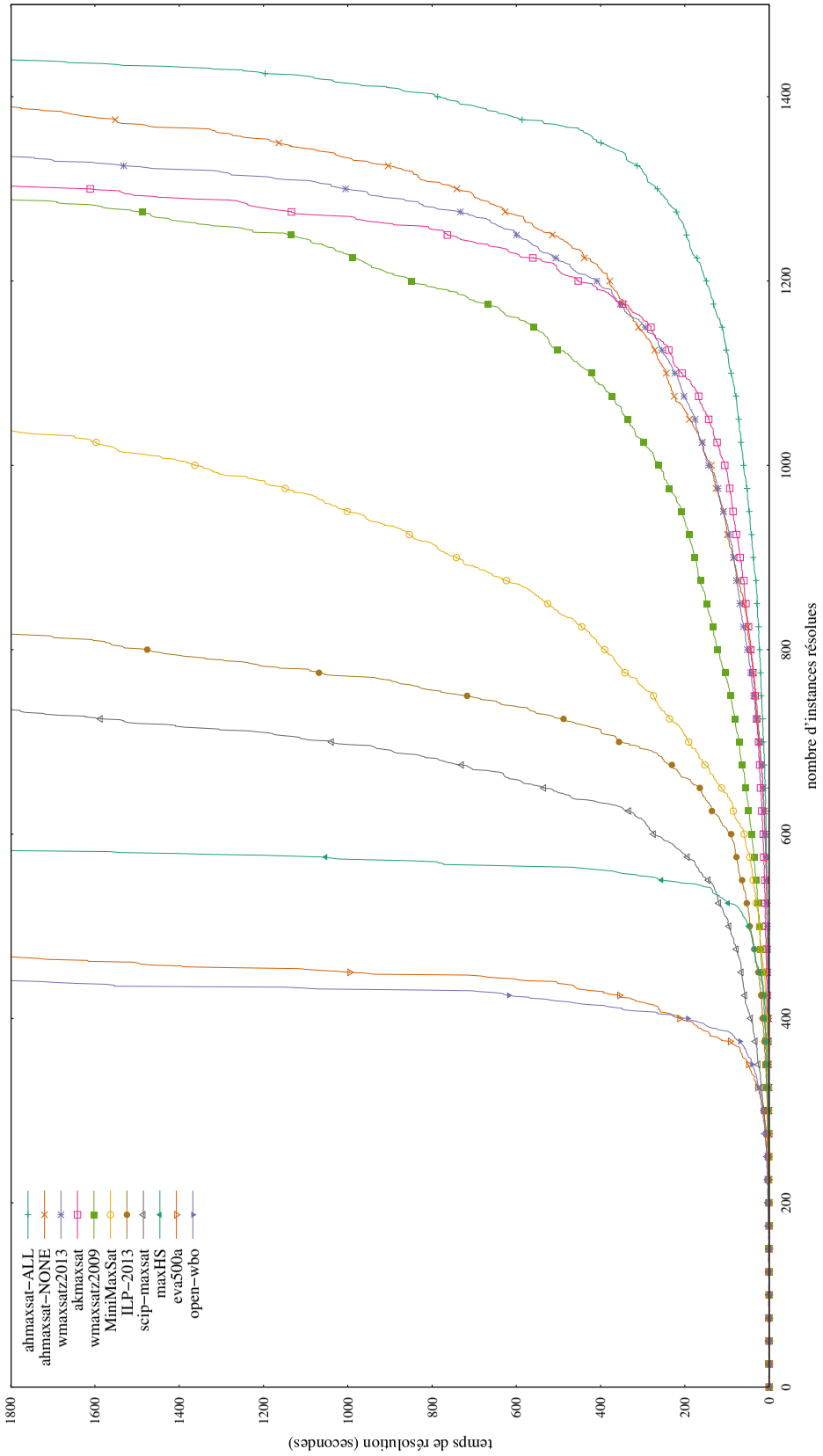


FIGURE 8.1 – Temps de résolution cumulés des solveurs sur le benchmark de la MSE 2014.

voir que les performances d'AHMAXSAT-ALL et d'AKMAXSAT sont proches, suivies par celles des autres solveurs de type séparation et évaluation. Sur les instances Max-SAT partiel (figure 8.4b, les solveurs basé sur la reformulation en ILP (SCIP-MAXSAT et ILP-2013) sont les plus performants, suivis de près par MAXHS (qui combine reformulation en ILP et appels itératifs à un solveur SAT) et MINIMAXSAT. Viennent ensuite les autres solveurs itératifs SAT et les solveurs de type séparation et évaluation AHMAXSAT-ALL et WMAXSATZ2013. Les autres solveurs de type séparation et évaluation sont peu performants sur ces instances. Enfin, les instances Max-SAT crafted valué et partiel valué (figure 8.4c) sont également dominées par ILP-2013 et MAXHS. Notre solveur AHMAXSAT-ALL arrive en troisième position, suivi de SCIP-MAXSAT, AHMAXSAT-NONE et MINIMAXSAT. Les autres solveurs de type séparation et évaluation et les solveurs SAT itératifs sont nettement moins efficaces sur les instances considérées.

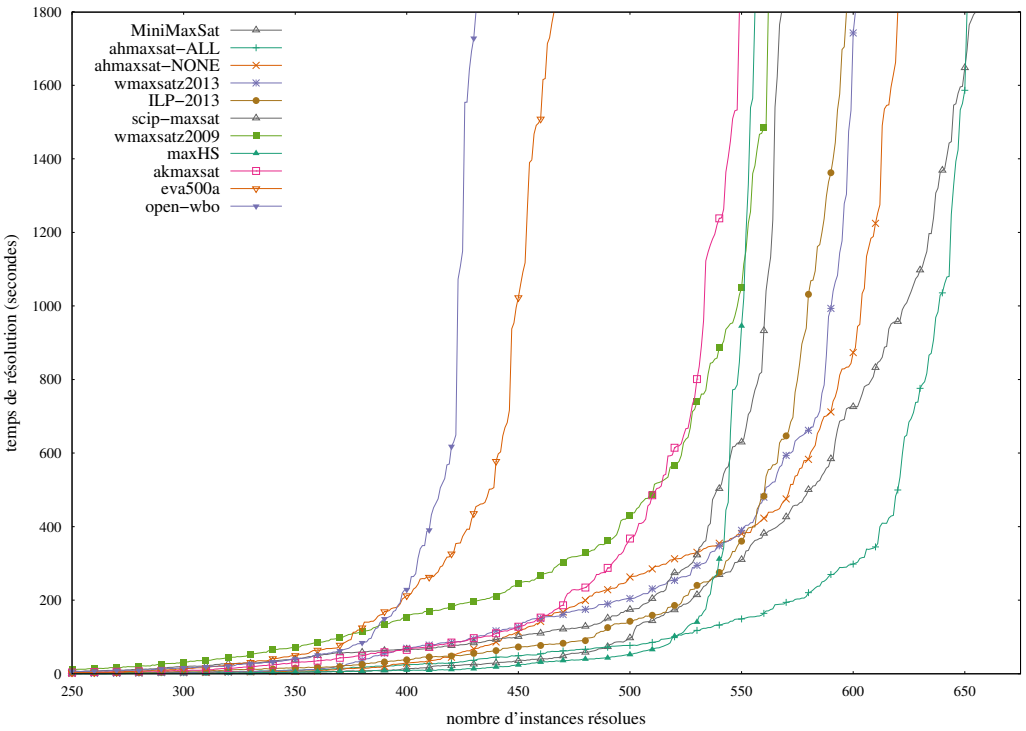
Les instances aléatoires sont dominées par les solveurs de type séparation et évaluation (figure 8.2b). Notre solveur est le plus performant, suivi de AKMAXSAT, WMAXSATZ 2013 et WMAXSATZ 2009. On peut observer que les performances de MINIMAXSAT sur ces instances sont nettement en retrait en comparaison de celles des autres solveurs de type séparation et évaluation. Si l'on considère les résultats sur les instances Max-SAT uniquement (non-partiel, non-valué, figure 8.3a) on peut voir que les performances d'AHMAXSAT-ALL, AHMAXSAT-NONE, AKMAXSAT et WMAXSATZ2013 sont très proches, avec un léger avantage pour AHMAXSAT-ALL. Sur les instances Max-SAT partiel (figure 8.3b), AHMAXSAT-ALL est sensiblement plus performant que les autres solveurs de type séparation et évaluation. Enfin, sur les instances Max-SAT valué et Max-SAT partiel valué (figure 8.3c), les performances d'AHMAXSAT-ALL et d'AKMAXSAT sont très proches, les autres solveurs de type séparation et évaluation étant légèrement en retrait.

Si l'on considère séparément les instances Max-SAT valué et Max-SAT partiel valué (figure 8.5), on peut voir que les performances des solveurs diffèrent entre ces classes d'instances. Les solveurs basés sur la reformulation en ILP, SCIP-MAXSAT et ILP-2013, ne résolvent quasiment aucune instance sur les instances aléatoire Max-SAT valué non-partiel tandis qu'ils sont plus performants sur les instances aléatoires Max-SAT partiel valué (bien que nettement inférieures aux solveurs de type séparation et évaluation). De la même manière, les instances *crafted* Max-SAT partiel valué sont dominées par les solveurs basés sur la reformulation en ILP, tandis que les instances *crafted* Max-SAT valué non-partiel sont dominées par les solveurs de type séparation et évaluation.

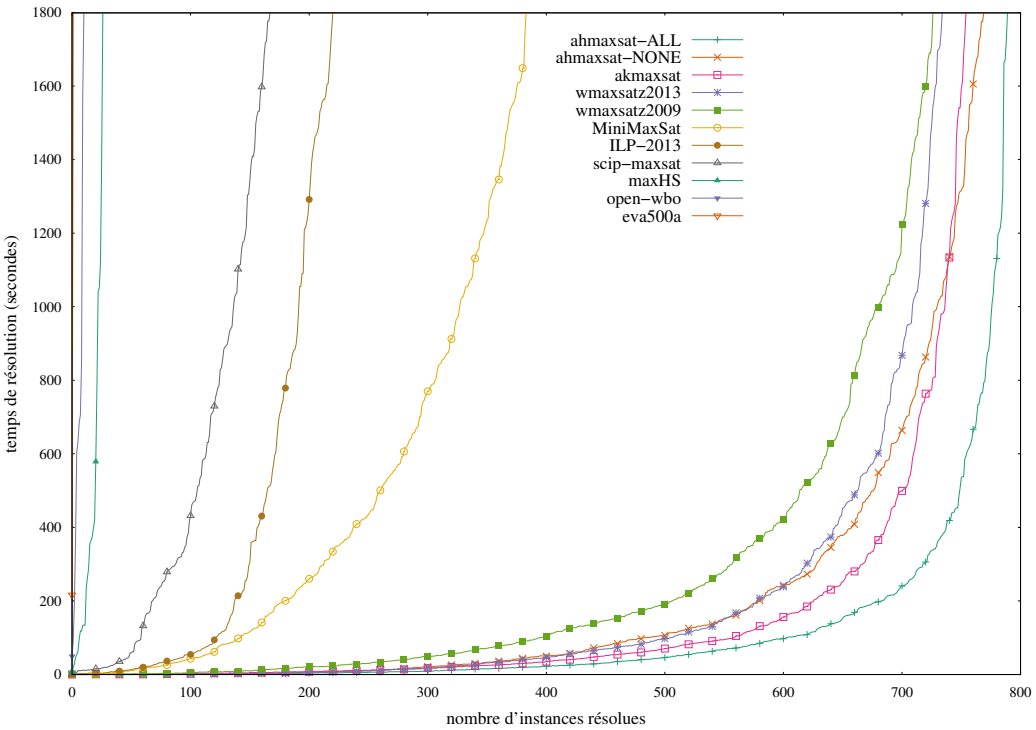
Ces résultats sont concordants avec ceux de la MSE 2014, où notre solveur était classé premier dans 3 des 9 catégories d'instances et second dans une quatrième. Les solveurs de type séparation et évaluation dominent les autres sur les instances aléatoires et *crafted* non-partielle, les solveurs basés sur la reformulation en ILP dominent les instances *crafted* partielles tandis que les solveurs basés sur l'appel itératif à un solveur SAT dominent les instances industrielles (non testées dans cette évaluation).

8.2.2 Comparaison des solveurs de type séparation et évaluation

Si on limite la comparaison aux solveurs de type séparation et évaluation, on peut voir (table 8.1) que les deux variantes d'AHMAXSAT résolvent davantage d'instances que n'importe lequel des autres solveurs. Après eux, le solveur le plus performant est WMAXSATZ2013 qui résout 105 instances de moins qu'AHMAXSAT-ALL. Puis viennent AKMAXSAT, WMAXSATZ2009 et MINIMAXSAT avec respectivement 137, 151 et 402 instances

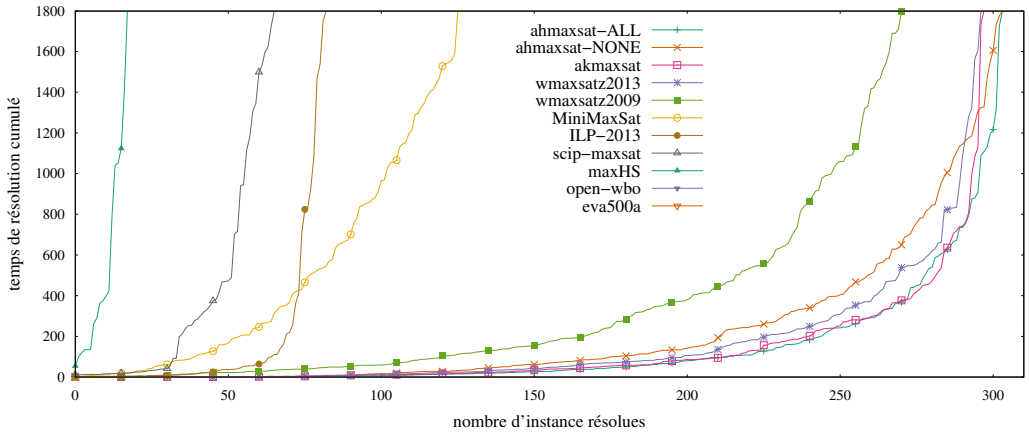


(a) Instances *crafted*.

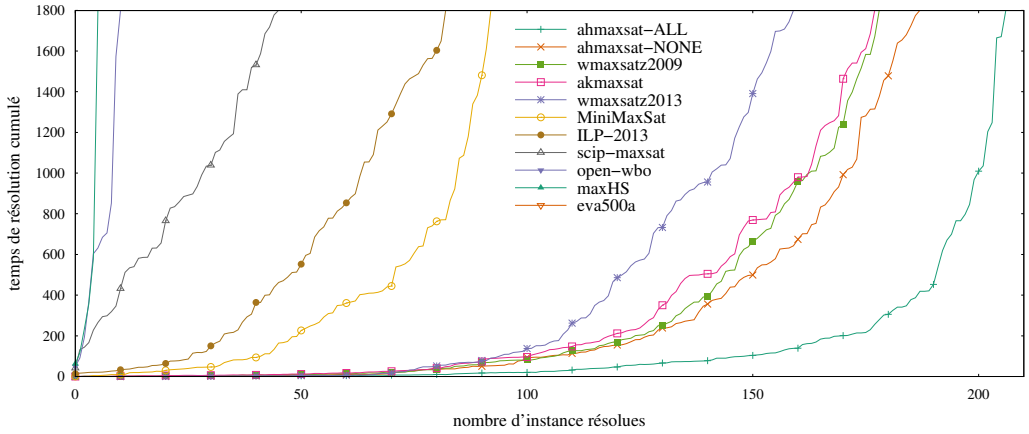


(b) Instances aléatoires.

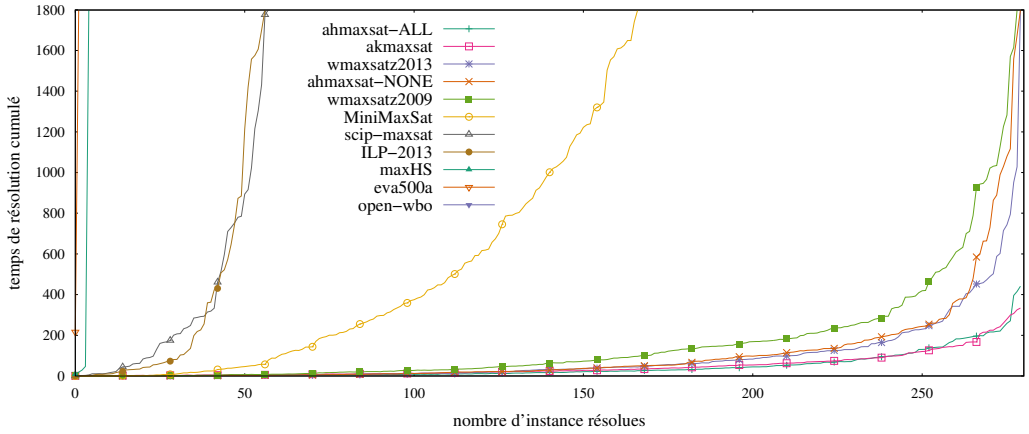
FIGURE 8.2 – Détails des temps de résolution cumulés sur les instances aléatoires et *crafted* de la MSE 2014.



(a) ms_random

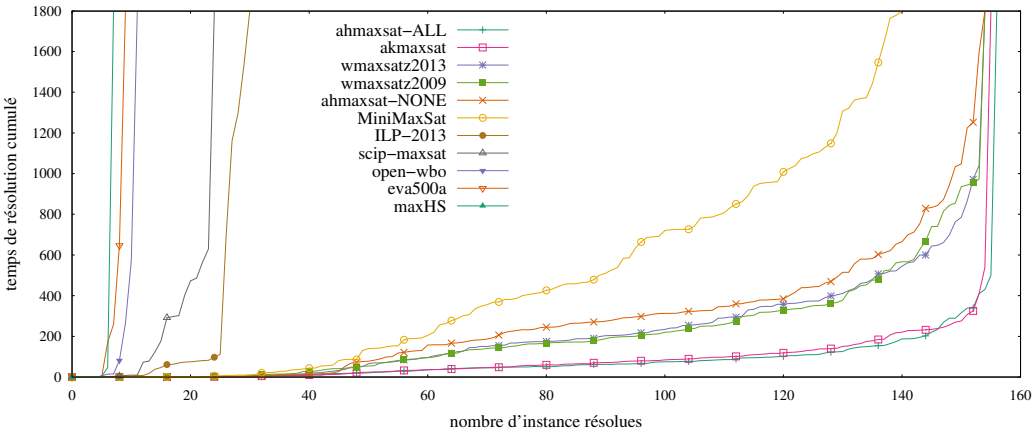


(b) pms_random

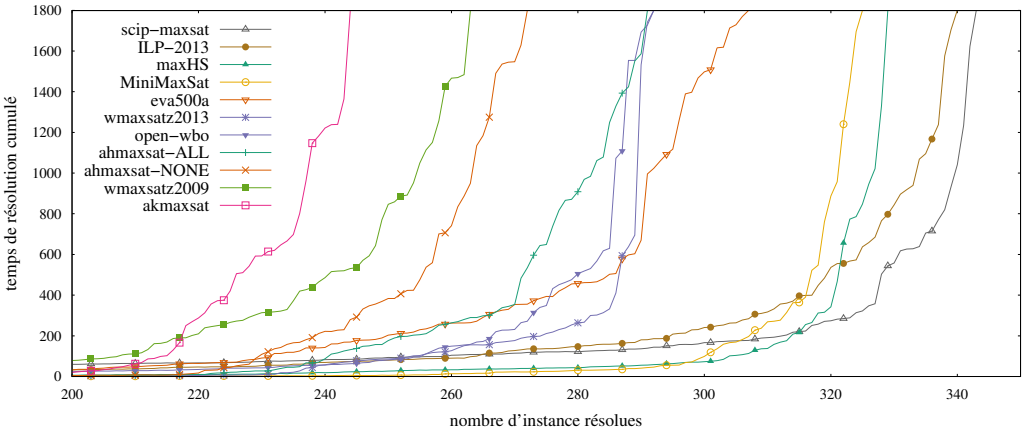


(c) wms+wpms_random

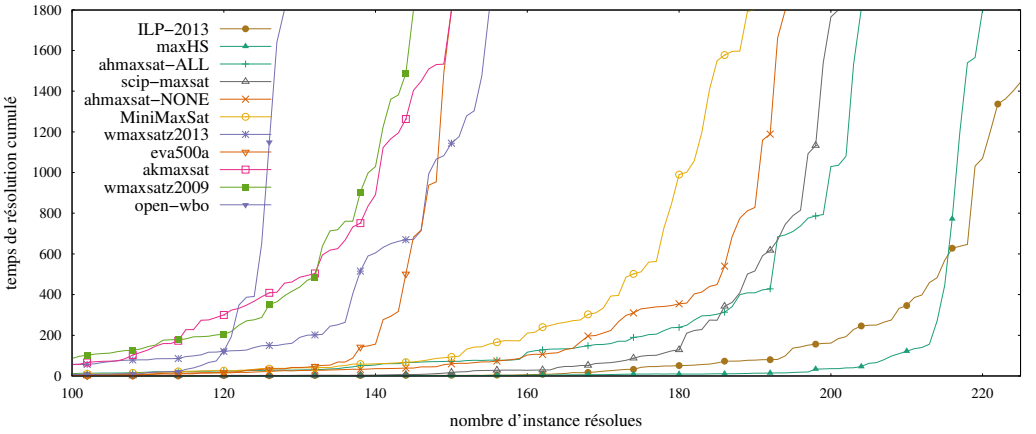
FIGURE 8.3 – Temps de résolution cumulés des solveurs sur le benchmark de la MSE 2014.



(a) ms_crafted

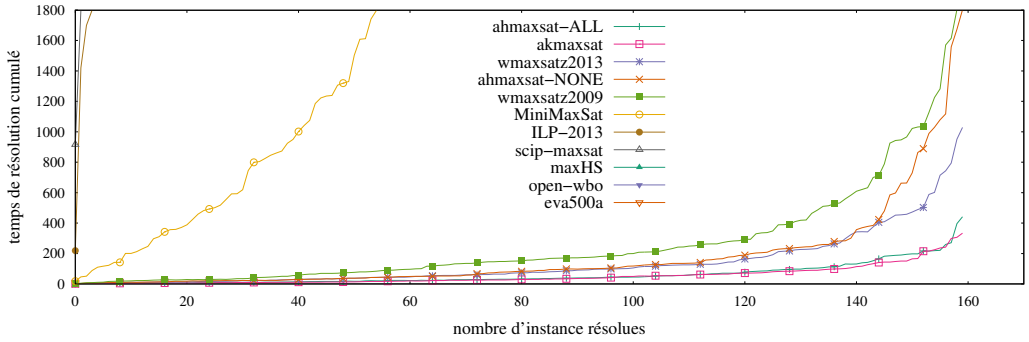


(b) pms_crafted

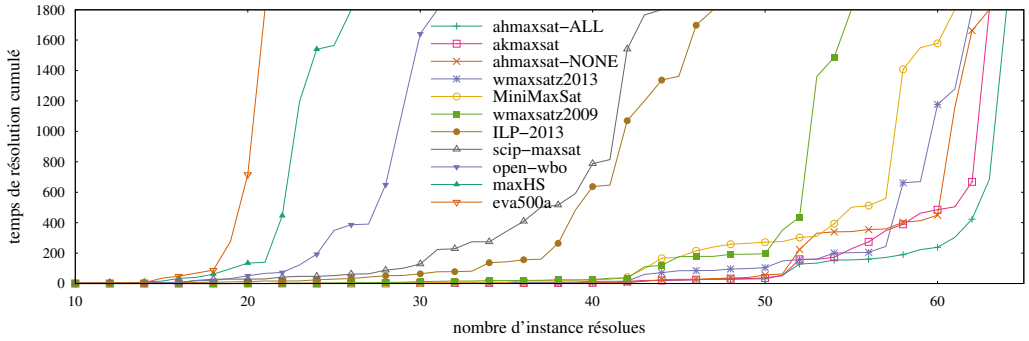


(c) wms+wpms_crafted

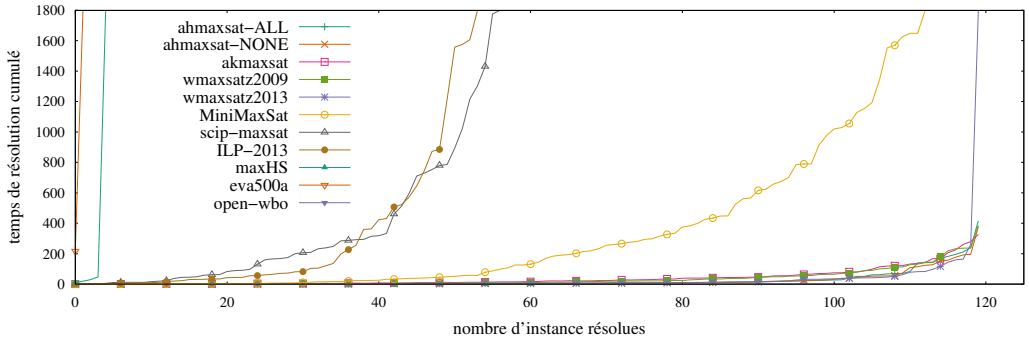
FIGURE 8.4 – Temps de résolution cumulés des solveurs sur le benchmark de la MSE 2014.



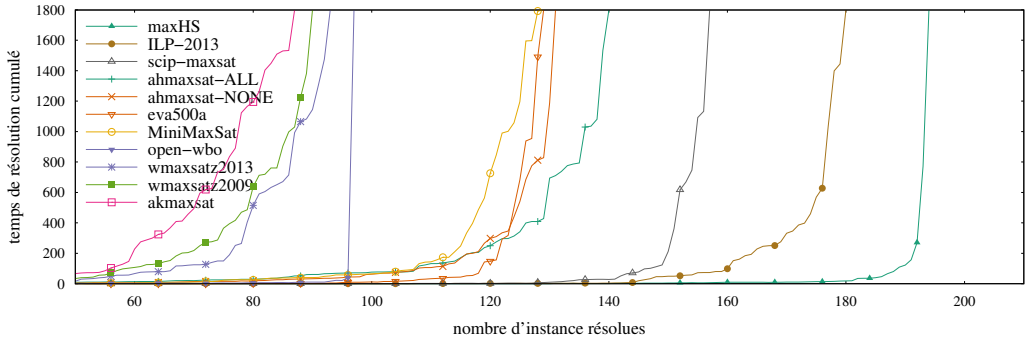
(a) wms_random



(b) wms_crafted



(c) wpms_random



(d) wpms_crafted

FIGURE 8.5 – Temps de résolution cumulés des solveurs sur le benchamrk de la MSE 2014.

résolues de moins qu'AHMAXSAT-ALL. Si l'on considère le temps moyen de résolution sur l'ensemble du benchmark, on peut voir qu'AHMAXSAT-NONE, WMAXSATZ2013, AKMAXSAT, WMAXSATZ2009 et MINIMAXSAT sont respectivement en moyenne 70%, 41%, 23%, 104% et 157% plus lent qu'AHMAXSAT-ALL.

On peut observer certaines particularités parmi ces solveurs. WMAXSATZ2009 est globalement efficace, sans dominer les autres solveurs sur aucune des catégories d'instances. WMAXSATZ2013 est lui aussi équilibré. Il domine les autres solveurs sur les instances max-3-sat aléatoires partielles et ses performances sont honorables sur les instances *crafted*. AKMAXSAT est efficace sur les instances aléatoires non-partielles, et en particulier sur les max-3-sat. Enfin, MINIMAXSAT est performant sur les instances structurées et en particulier sur les Max-SAT partiel, mais il n'est pas compétitif sur les instances aléatoires.

Pour renforcer et étendre les comparaisons entre certains types de solveurs, nous avons également généré des instances *random* (max-2-sat et max-3-sat, valuées et non-valuées) et *crafted* (maxcut) supplémentaires [4] [5].

Nous avons progressivement fait croître les nombres de variables et de clauses des instances de la MSE 2014 jusqu'à atteindre les limites des solveurs actuels (quand plus aucune instance n'est résolue par les solveurs). Nous avons obtenu 3000 instances *random*, avec des nombres de variables et de clauses variant respectivement de 120 à 200 et de 1200 à 2600 pour les max-2-sat et de 70 à 110 et de 700 à 1500 pour les max-3-sat. Les nombres de variables et de clauses des 1350 nouvelles instances *crafted* varient respectivement de 140 à 220 et de 1200 à 2600. Nous n'avons pas trouvé de générateur produisant des instances partielles.

TABLE 8.2 – Résultats des solveurs complets sur les instances générées. Le nombre d'instances résolues par chaque solveur est indiqué, avec entre parenthèses le temps moyen de résolution.

	Instance classes	#	AHMAXSAT-ALL	AKMAXSAT	WMAXSATZ 2013	AHMAXSAT-NONE	WMAXSATZ 2009	MINIMAXSAT
ms	crafted	1350	**412(457,4)**	405(516,3)	236(670,2)	206(707,8)	248(649,1)	146(766,1)
ms	random/max2sat	1150	**679(412,7)**	644(477,5)	540(477,5)	461(524,4)	431(547,5)	11(949,1)
ms	random/max3sat	350	224(505,4)	**237(508,7)**	217(478,7)	203(520,7)	184(554,1)	88(712,3)
wms	random/max2sat	1150	**921(340,9)**	857(339,2)	673(422,9)	588(515,7)	528(502,5)	60(959,2)
wms	random/max3sat	350	236(476,1)	**266(411,8)**	203(425,7)	236(467,9)	174(397,0)	97(596,8)
	Overall	4350	**2472(402,4)**	2409(424,8)	1869(469,8)	1694(527,4)	1565(523,6)	402(724,8)

Les résultats obtenus sur ces nouvelles instances sont présentés dans la table 8.2. Sur l'ensemble des instances, notre solveur AHMAXSAT-ALL est le plus efficace, avec 2472 instances résolues sur 4350. Il est suivi par AKMAXSAT qui résout 63 instances de moins. Puis viennent WMAXSATZ2013, AHMAXSAT-NONE, WMAXSATZ2009 et MINIMAXSAT avec respectivement 603, 778, 907 et 2070 instances résolues de moins qu'AHMAX-

4. Les générateurs sont disponibles à l'adresse `http://web.udl.es/usuaris/m4372594/jair-generators`

5. Les instances générées sont téléchargeables à l'adresse `http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html`

SAT-ALL. On peut observer qu'AHMAXSAT-ALL est significativement plus performant que les autres solveurs sur les instances random/max2sat (non-valuées et valuées) et sur les crafted/maxcut. Cependant, il résout moins d'instances random/max3sat qu'AKMAXSAT.

La figure 8.6 montre le temps de résolution cumulé de chacun des solveurs testés. On voit que notre solveur est suivi de près par AKMAXSAT, tandis que WMAXSATZ2013, AHMAXSAT-NONE, WMAXSATZ2009 sont significativment moins efficaces. Cela confirme également les faibles performances de MINIMAXSAT sur les instances aléatoires.

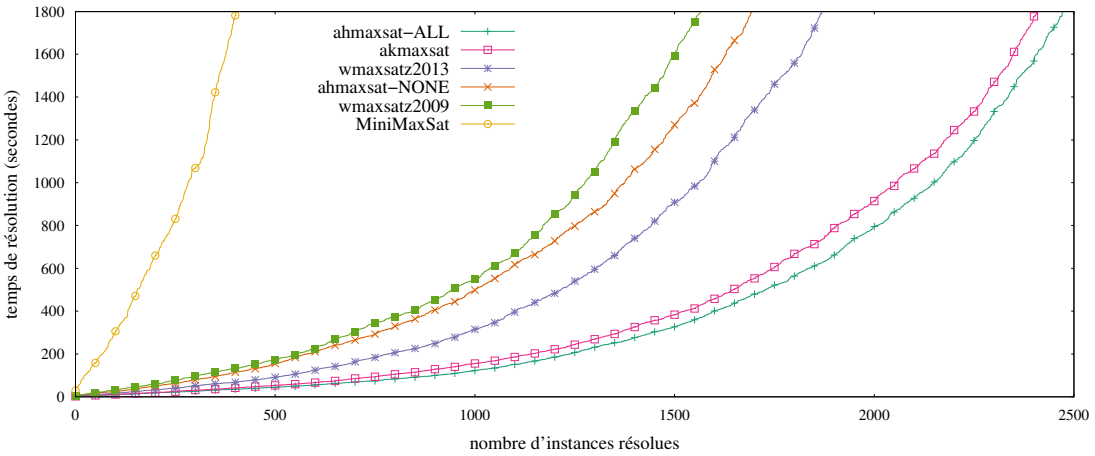


FIGURE 8.6 – Temps de résolution cumulés des solveurs de type séparation et évaluation sur les instances générées.

8.3 Conclusion

Nous avons donné dans ce chapitre une vue d'ensemble de notre solveur AHMAXSAT. Nous avons décrit l'heuristique de branchement, les règles d'inférences utilisées pour étendre l'interprétation courante et la fonction de calcul de la borne inférieure. Nous avons également discuté des interactions entre les différentes contributions présentées dans les chapitres précédents. Enfin, nous avons comparé AHMAXSAT aux solveurs séparation les plus récents et à certains des meilleurs solveurs complets de l'état de l'art.

Les résultats obtenus confirment ceux de la *Max-SAT Evaluation 2014* et montrent qu'AHMAXSAT est compétitif avec les solveurs de l'état de l'art. Il domine les autres solveurs de type séparation et évaluation sur la majorité des instances considérées et les autres solveurs complets sur les instances faiblement structurées.

Une meilleure prise en compte de la structure des instances sera probablement un élément déterminant pour rendre plus compétitifs les solveurs de type séparation et évaluation sur les instances industrielles et certaines *crafted*. Une autre piste de recherche prometteuse concerne l'intégration de techniques de résolution propres à SAT, qui pourra permettre d'améliorer l'efficacité des solveurs sur les instances Max-SAT partiel.

Chapitre 9

Règles d'inférence et recherche locale pour Max-SAT

Les méthodes de recherche locale, présentées dans les sections 1.3.2 et 2.4.1, parcourent l'espace de recherche de manière non exhaustive. En partant d'une interprétation complète choisie aléatoirement, elles cherchent à atteindre la meilleure solution possible (i.e. qui minimise la somme des poids des clauses falsifiées) par une série de mouvements (les *flips*).

Nous avons vu dans le chapitre 2 les règles d'inférence de la propagation unitaire et de la max-résolution, qui peuvent être combinées pour détecter et transformer les sous-ensembles inconsistants de la formule. Lorsqu'une clause vide est détectée par propagation unitaire, l'analyse des étapes de propagation unitaire permet de construire un sous-ensemble inconsistant de la formule. Celui-ci peut ensuite être transformé par la règle de la max-résolution. Après transformation, une clause vide (la résolvante) est ajoutée à la formule et l'inconsistance est en quelque sorte "isolée" du reste de la formule. Cette méthode est largement utilisée lors du calcul de la borne inférieure dans les solveurs séparation et évaluation pour Max-SAT (cf. section 2.3.4) et nous avons étudié certains de ses aspects dans les chapitres 3 à 7 de cette thèse.

Nous proposons dans ce chapitre d'intégrer la méthode de détection et de transformation des sous-ensembles inconsistants dans le solveur de recherche locale pour Max-SAT AdaptNovelty++. L'objectif est de simplifier la formule en entrée en identifiant et en transformant certains de ses sous-ensembles inconsistants pour permettre à

la recherche locale de trouver plus rapidement une solution de bonne qualité. La principale difficulté à surmonter est le maintien d'un graphe d'implications cohérent pour rendre possible la construction de sous-ensembles inconsistants dans le contexte de la recherche locale (ce point à déjà été étudié dans le contexte de SAT, par exemple dans [LSB04, HK05, ALMS09, ALMS10, GH11, HT12]). L'application des règles d'inférence étant coûteuse en temps de calcul, il faut également trouver un juste milieu entre le temps passé à appliquer les règles d'inférence et le temps passé à résoudre l'instance. Sinon, l'algorithme peut s'avérer inefficace en pratique. Nous nommons le solveur obtenu IRAnovelty++.

Ce chapitre est organisé comme suit. Nous rappelons le fonctionnement de l'algorithme de recherche locale AdaptNovelty++ dans la section 9.1 avant de présenter dans la section 9.2 la manière dont nous lui avons intégré la méthode de détection et de transformation des sous-ensembles inconsistants, formant ainsi l'algorithme IRAnovelty++. Nous présentons dans la section 9.3 les résultats de l'optimisation empirique des paramètres d'AdaptNovelty++ et d'IRAnovelty++ avant de comparer expérimentalement ces solveurs à plusieurs des meilleurs solveurs de recherche locale de l'état de l'art. Nous concluons dans la section 9.4. Les travaux présentés dans ce chapitre ont été publiés dans des conférences internationale [AH12] et nationale [AH13].

9.1 Algorithme de recherche locale : AdaptNovelty++

Nous présentons dans cette section l'algorithme de recherche locale AdaptNovelty++ [LH05b, Hoo02], qui est une variante améliorée de WalkSat (cf. section 1.3.2.3). L'algorithme 9.1 donne le schéma général d'un algorithme de recherche locale pour SAT basé sur WalkSat adapté pour Max-SAT. Il commence par initialiser I_{OPT} et OPT qui servent à mémoriser respectivement la meilleure solution rencontrée par l'algorithme et la somme des poids des clauses qu'elle falsifie (lignes 2-3). Puis, l'algorithme génère aléatoirement une interprétation complète I (fonction `interprétation_complète_aléatoire`, ligne 5). Itérativement, il met à jour OPT et I_{OPT} si la somme des poids des clauses falsifiées par I est inférieure à la meilleure solution trouvée jusqu'à présent (lignes 7-9). Puis, il choisit aléatoirement une clause falsifiée c et une variable $x_i \in c$ qu'il flippe (lignes 10-12). Après *MaxFlips* flips, l'algorithme revient à la ligne 5 et génère une nouvelle interprétation complète aléatoire. Ce processus est répété pour un nombre donné de relances (*MaxRelances*). L'algorithme renvoi (OPT, I_{OPT}) si I_{OPT} satisfait toutes les clauses dures de la formule et INCONNU sinon (lignes 13-14).

Les variantes de WalkSat diffèrent principalement par leur heuristique de choix des variables à flipper [MSK97]. Ces heuristiques se basent généralement sur deux compteurs : `break` et `make` qui calculent, pour une variable x_i, le nombre de clauses qui seront respectivement falsifiées et satisfaites si x_i est flippée. La différence entre `break`(x_i) et `make`(x_i) est notée `score`(x_i). Certaines heuristiques considèrent également le nombre de flips réalisés depuis la dernière modification de la valeur des variables, qui est donné par la fonction `age()`.

Algorithme 9.1 : Schéma général d'un algorithme de type WalkSat pour Max-SAT partiel valué.

Data : Une formule Φ et deux paramètres *MaxRelances* et *MaxFlips*.
Result : Un couple (OPT, I_{OPT}) avec I_{OPT} la meilleure solution atteinte par l'algorithme et OPT la somme des poids des clauses falsifiées par cette interprétation ou INCONNU si aucune solution de la partie dure n'a été trouvée.

```
1  begin
2      OPT ← ∑_{c∈Φ} poids(c);
3      I_OPT ← ∅;
4      for i ← 1 to MaxRelances do
5          I ←interprétation_complète_aléatoire();
6          for j ← 1 to MaxFlips do
7              if ∑_{c∈Φ, c|_I=□} poids(c) < OPT then
8                  OPT ← ∑_{c∈Φ, c|_I=□} poids(c);
9                  I_OPT ← I;
10             c ← choix_clause_falsifiée(Φ|_I);
11             x ← choix_variable_à_flipper(c);
12             I ← flip(I, x);
13     if OPT ≤ ∑_{c∈Φ_S} poids(c) then return (OPT, I_OPT) ;
14     else return INCONNU ;
```

La version originale de WalkSat (algorithme 9.2), une fois la clause falsifiée c sélectionnée, choisit la variable à flipper aléatoirement parmi les variables de c qui ont une valeur de break nulle s'il en existe. Sinon, elle choisit aléatoirement une variable de c avec une probabilité p et avec une probabilité $1-p$ elle choisit la variable avec la plus petite valeur de break.

Algorithme 9.2 : Algorithme WalkSAT : fonction `choix_variable_à_flipper`

Data : Une clause c falsifiée par l'interprétation courante.
Result : Une variable x à flipper.

```
1  begin
2      if {x ∈ var(c) t.q. break(x) = 0} ≠ ∅ then
3          return une variable choisie aléatoirement dans {x ∈ var(c) t.q. break(x) = 0};
4      else
5          with probabilité p do
6              return une variable choisie aléatoirement dans var(c);
7          otherwise
8              return une variable choisie aléatoirement dans
               {x ∈ var(c) t.q. ∀x' ∈ var(c), break(x) ≤ break(x')};
```

Dans Novelty [MSK97] (algorithme 9.3), les variables apparaissant dans c sont triées en fonction de leurs scores. L'algorithme considère les deux variables de plus hauts scores (lignes 2-3). Si la variable de plus haut score n'est pas celle qui, des variables de c, a été flippée le plus récemment alors elle est choisie (ligne 4). Sinon, avec une probabilité p,

l'algorithme choisit celle de second plus haut score et avec une probabilité $1-p$ celle de plus haut score (lignes 5-6).

Algorithme 9.3 : Algorithme Novelty : fonction `choix_variable_à_flipper`

Data : Une clause c falsifiée par l'interprétation courante.
Result : Une variable x à flipper.

```
1 begin
2     x_i ← variable de plus fort score de var(c);
3     x_j ← variable de second plus fort score de var(c);
4     if ∃x_k ∈ var(c) s.t. age(x_k) < age(x_i) then return x_i ;
5     else
6         with probabilité p do return x_j ;
7         otherwise return x_i ;
```

Dans [LH05b], Novelty est étendue à Novelty++ comme suit (algorithme 9.4) : avec une probabilité dp (*diversification probability*) l'algorithme choisit la variable qui a été flippée le moins récemment (lignes 2-3) et avec une probabilité $1-dp$ il fait comme Novelty (lignes 5-9).

Algorithme 9.4 : Algorithme Novelty++ : fonction `choix_variable_à_flipper`

Data : Une clause c falsifiée par l'interprétation courante.
Result : Une variable x à flipper.

```
1  begin
2      with probabilité dp do
3          return la variable x_k ∈ var(c) t.q. ∀x_l ∈ var(c), age(x_k) > age(x_l);
4      otherwise
5          x_i ← variable de plus fort score de var(c);
6          x_j ← variable de second plus fort score de var(c);
7          if ∃x_k ∈ var(c) s.t. age(x_k) < age(x_i) then return x_i ;
8          else
9              with probabilité p do return x_j ;
10             otherwise return x_i ;
```

Les valeurs optimales de p et dp dépendent de l'instance traitée et sont difficiles à déterminer. Hoos propose dans [Hoo02] un mécanisme d'ajustement dynamique de la valeur de p en fonction de l'évolution de la recherche. Ce mécanisme a été étendu par Li et Huang à la probabilité dp [LH05b]. Si le nombre de clauses falsifiées n'a pas été réduit pendant un certain nombre de flips, alors les valeurs de p et dp sont augmentées pour favoriser la diversification. À l'inverse, ces valeurs sont réduites lorsque le nombre de clauses falsifiées diminue. La combinaison de Novelty++ et du mécanisme d'adaptation des probabilités est appelée AdaptNovelty++.

Nous avons ajouté à AdaptNovelty++ une heuristique de sélection des clauses falsifiées pour les formules valuées. Plutôt que de choisir aléatoirement parmi l'ensemble des clauses falsifiées de la formule, AdaptNovelty++ choisit parmi les clauses falsifiées de plus fort poids. Pour éviter de sélectionner toujours les mêmes clauses, nous avons

ajouté un mécanisme de tabou. Lorsqu'une clause est choisie elle devient tabou, c'est-à-dire qu'elle ne peut plus être sélectionnée. La durée du tabou est fixée empiriquement à 0,9 fois le nombre de clauses falsifiées de la formule.

9.2 IRANOVELTY++ : recherche locale et règles d'inférence pour Max-SAT

Notre objectif est d'intégrer des règles d'inférence dans un algorithme de recherche locale pour améliorer ses performances lorsqu'il est appliqué au problème Max-SAT. Nous avons donc ajouté à ADAPTNOVELTY++ les éléments nécessaires à la détection et à la transformation des sous-ensembles inconsistants, formant ainsi un nouveau solveur incomplet pour Max-SAT : IRANOVELTY++.

La première difficulté à surmonter est l'intégration de la propagation unitaire simulée (qui est nécessaire pour la détection des sous-ensembles inconsistant) dans ADAPTNOVELTY++. Comme les algorithmes de recherche locale travaillent sur des interprétations complètes tandis que la propagation unitaire simulée travaille sur une interprétation partielle, notre solveur maintient simultanément deux affectations : une complète pour la recherche locale (I_{RL}) et une partielle pour la détection des conflits par SUP (I_{SUP}). Cette seconde interprétation est remplie par les flips réalisés par la partie recherche locale de l'algorithme. Le maintient du graphe d'implications, qui mémorise les étapes de propagation unitaire, est plus complexe que dans le contexte de SAT pour deux raisons : (1) les affectations de I_{SUP} ne sont pas faites et défaites chronologiquement et (2) l'application de la règle d'inférence supprime des clauses de la formule, clauses qui peuvent être des sources de propagation. Pour permettre la construction et le maintien d'un graphe d'implications cohérent dans ce contexte, IRANOVELTY++ utilise le schéma de propagation MPS présenté dans le chapitre 3 : il considère toutes les sources de propagation des variables. Les flips réalisés par la recherche locale ne permettent pas nécessairement de construire un graphe d'implications valide. Nous autorisons donc la suppression d'une affectation l de I_{SUP} dans deux situations : si $\bar{l}$ est ajouté à I_{SUP} ou si elle provoque un conflit direct avec des affectations plus récentes (i.e. s'il existe une clause $\{\bar{l}, \bar{l}_1, \ldots \bar{l}_k\}$ telle que $l_1, \ldots, l_k$ appartiennent à I_{SUP} et sont tous plus récents que l.

La propagation unitaire simulée est ensuite appliquée sur la formule simplifiée par l'interprétation I_{SUP}. Lorsqu'un conflit est détecté (i.e. quand il existe une clause c falsifiée par les affectations déduites par propagation unitaire), le graphe d'implications plein est analysé pour construire un sous-ensemble inconsistant. Ce dernier est ensuite transformé par plusieurs étapes de max-résolution appliquées dans l'ordre inverse des propagations. La clause résolvante ajoutée pour chaque sous-ensemble inconsistant transformé est directement falsifiée par l'interprétation I_{SUP}. Ces étapes sont similaires à celles réalisées par les solveurs séparation et évaluation lors du calcul de la borne inférieure. On peut noter cependant qu'IRANOVELTY++ n'utilise ni l'heuristique SIS de construction des sous-ensembles inconsistants (cf. chapitre 3) ni l'heuristique SIR de choix de l'ordre d'application des étapes de max-résolution (cf. chapitre 4) qui ont été développées après les travaux présentés dans ce chapitre.

La seconde difficulté inhérente à l'application des règles d'inférence concerne le contrôle de ses deux principaux inconvénients : (1) le temps de calcul nécessaire à son application et (2) l'augmentation de la taille de la formule. Nous avons défini deux li-

mites pour contrôler les transformations appliquées à la formule et limiter sa taille après transformation : *LimSUP* et *LimMR*.

— *LimSUP* définit la taille maximum que peut atteindre la formule transformée, exprimée en ratio de la taille de l'originale. Lorsque cette limite est atteinte, SUP n'est plus appliquée (et par conséquent la max-résolution non plus) et seule la partie recherche locale de l'algorithme reste active. Avec des valeurs < 1, la formule n'est jamais modifiée et l'algorithme se comporte comme ADAPTNOVELTY++.

— *LimMR* définit la taille maximum autorisée des résolvants lors de l'application de la max-résolution, exprimée en ratio de la taille de la plus petite clause du sous-ensemble inconsistant. Lorsque cette limite est dépassée, les modifications ne sont pas appliquées et le conflit est simplement ignoré.

Comme ADAPTNOVELTY++, IRANOVELTY++ relance la recherche locale avec une nouvelle interprétation complète choisie aléatoirement après $MaxFlips*|\Phi|$ flips. Il inclut également le mécanisme d'adaptation des probabilités de diversification et l'heuristique de choix des clauses falsifiées basée sur le poids des clauses, tous deux présentés dans la section 9.1.

Algorithme 9.5 : Schéma général de l'algorithme IRANOVELTY++.

Data : Φ, *MaxRelances*, *MaxFlips*, *LimMR* and *LimSUP*.
Result : Un couple (OPT, I_{OPT}) avec I_{OPT} la meilleure solution atteinte par l'algorithme et OPT la somme des poids des clauses falsifiées par I_{OPT} ou INCONNU si aucune solution de la partie dure n'a été trouvée.

begin
  $I_{OPT} \leftarrow \emptyset$; $OPT \leftarrow \sum_{c\in\Phi} poids(c)$;
  $\Phi' \leftarrow \Phi$; $I_{SUP} \leftarrow \emptyset$;
  for $j \leftarrow 1$ **to** *MaxRelances* **do**
    $I_{RL} \leftarrow$ `interprétation_complète_aléatoire()`;
    for $k \leftarrow 1$ **to** $MaxFlips*|\Phi|$ **do**
      if $\sum_{c\in\Phi',\ c|_{I_{RL}}=\square} poids(c) < OPT$ **then**
        $OPT \leftarrow \sum_{c\in\Phi',\ c|_I=\square} poids(c)$;
        $I_{OPT} \leftarrow I_{RL}$;
      $c \leftarrow$ `choix_clause_falsifiée(`$\Phi|_{I_{RL}}$`)`;
      $x \leftarrow$ `choix_variable_à_flipper(`c`)`;
      $I_{RL} \leftarrow$ `flip(`I_{RL}, x`)`;
      $I_{SUP} \leftarrow$ `affecte(`I_{SUP}, l`)`;
      if $|\Phi'| <$ *LimSUP*$*|\Phi|$ **then**
        // l est le littéral correspondant à x modulo sa valeur dans I_{RL}
        $G = (V, A) \leftarrow$`SUP*(`$\Phi|_I$`)`;
        while $\square \in V$ **do**
          $G' = (V', A') \leftarrow$`conversion_graphe_implications_plein(`G`)`;
          $\psi \leftarrow$`construction_sous_ensemble_inconsistant(`Φ'`,`G'`)`;
          if $taille\_resolvant \leq LimMR*min\{|c|, c \in \psi\}$ **then**
            $\Phi' \leftarrow$`transformation_sous_ensemble_inconsistant(`Φ'`,`ψ`)`;
          $G = (V, A) \leftarrow$ `SUP*(`$\Phi|_I$`)`;
  if $OPT \leq \sum_{c\in\Phi_S} poids(c)$ **then return** (OPT, I_{OPT}) ;
  else return *INCONNU* ;

IRANOVELTY++ est détaillé dans l'algorithme 9.5. Il commence par initialiser OPT, I_{OPT} et I_{SUP} (lignes 2-3). Puis, pour *MaxRelances* étapes, IRANOVELTY++ agit comme suit. Il commence par initialiser aléatoirement l'interprétation complète I_{RL} (ligne 5). Pour *MaxFlips*$*|\Phi|$ étapes, il met à jour OPT et I_{OPT} le cas échéant (lignes 7-9) puis il choisit une clause falsifiée et une variable de cette clause qu'il flippe (lignes 10-12). Puis il ajoute l'affectation réalisée à l'interprétation I_{SUP} en résolvant les éventuelles contradictions comme décrit au début de cette section (fonction `affecte()`, ligne 13). Si la taille de la formule transformée Φ' ne dépasse par *LimSUP* fois la taille de la formule originale Φ, l'algorithme applique la propagation unitaire simulée (fonction `SUP*()`, ligne 16). Tant que des clauses vides sont générées par la propagation unitaire, IRANOVELTY++ transforme le graphe d'implications plein en un graphe d'implications classique (fonction `conversion_graphe_implications_plein()`, ligne 18) qu'il analyse pour construire un sous-ensemble ψ de la formule qui est inconsistant sous l'interprétation I_{SUP} (fonction `construction_sous_ensemble_inconsistant()`, ligne 19). Si la taille de la clause résolvante qui sera obtenue après transformation par max-résolution est inférieure ou égale à *LimMR* fois la taille de la plus petite clause de ψ, alors les clauses de ψ sont transformées par max-résolution (ligne 20-21). Ces étapes sont répétées tant que SUP permet de détecter des sous-ensembles inconsistants. L'algorithme renvoie (OPT, I_{OPT}) si I_{OPT} satisfait toutes les clauses dures de la formule et INCONNU sinon (lignes 23-24).

9.3 Résultats expérimentaux

Nous présentons dans cette section les résultats de l'étude expérimentale que nous avons conduite. Celle-ci est divisée en deux parties. Dans la première, nous optimisons empiriquement les paramètres d'ADAPTNOVELTY++ et d'IRANOVELTY++, tandis que la seconde partie est consacrée à la comparaison expérimentale des deux solveurs avec quelques algorithmes de recherche locale de l'état de l'art.

9.3.1 Optimisation des paramètres

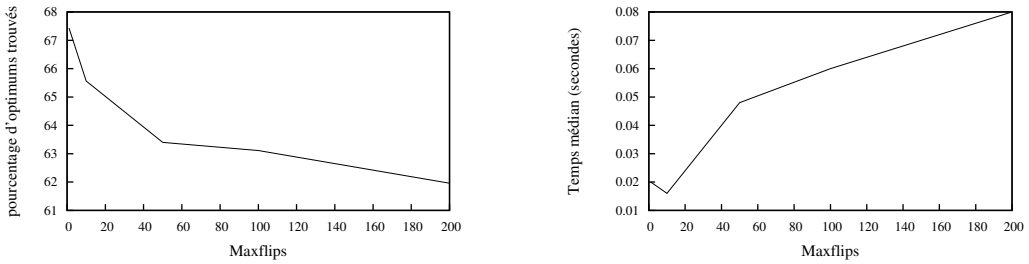
Les tests présentés dans cette section ont été réalisés sur un benchmark extrait de ceux des *Max-SAT evaluations* de 2011 et 2012. Nous avons choisi 694 instances, 150 instances non-valuées aléatoires Max-2-SAT, 150 non-valuées aléatoires Max-3-SAT, 167 non-valuées *crafted* Max-2-SAT, 80 valuées aléatoires Max-2-SAT, 80 valuées aléatoires Max-3-SAT et 65 valuées *crafted* Max-2-SAT. Ces instances sont représentatives des benchmarks utilisés lors des *Max-SAT evaluations* au moment de la publication des travaux présentés dans ce chapitre. Le temps d'exécution est fixé à 900 secondes et chaque variante des solveurs est lancée une fois sur chaque instance (un seul *run*). Les machines utilisées pour ces tests sont présentées dans la section 2.6.2.

Nous utiliserons comme critère de comparaison le pourcentage d'instances pour lesquelles l'optimum a été trouvé. Les valeurs des optima des instances ont été obtenues avec les solveurs complets présentés dans le chapitre 8. Lorsqu'aucun solveur complet n'est parvenu à résoudre une instance, nous utiliserons la meilleure valeur atteinte, que ce soit par un solveur complet ou par un des solveurs de recherche locale présentés dans la section 9.3.2. Quand deux solveurs (ou deux versions d'un même solveur) ont des résultats très proches sur ce premier critère, nous utilisons un critère additionnel : le temps

médian mis par le solveur pour atteindre l'optimum la première fois.

9.3.1.1 AdaptNovelty++

Nous avons tout d'abord étudié les performances d'AdaptNovelty++ en fonction du nombre de flips par relance (*MaxFlips*, exprimé comme un facteur de la taille de la formule originale). Le pourcentage d'optima trouvés (figure 9.1a) est légèrement plus élevé avec de faibles valeurs de *MaxFlips* (entre une et dix fois le nombre de clauses de la formule originale) et il décroît doucement lorsque cette valeur augmente. Le temps médian pour trouver la première fois l'optimum (figure 9.1b) confirme cette observation.



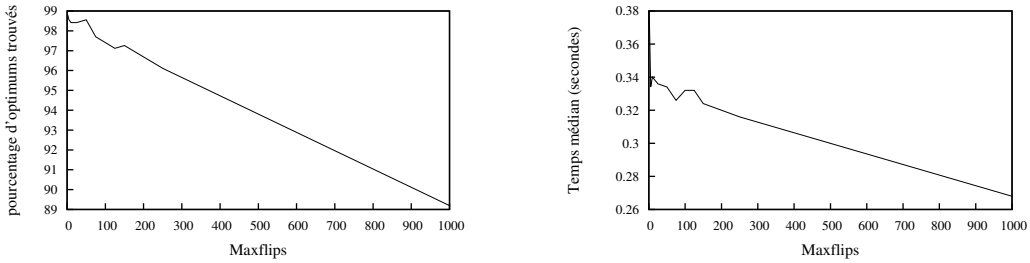
(a) Pourcentage d'optima trouvés (b) Temps médian pour atteindre l'optimum

Figure 9.1 – Impact de la valeur de *MaxFlips* sur AdaptNovelty++.

9.3.1.2 IRAnovelty++

Nous avons étudié individuellement chaque paramètre d'IRAnovelty++ : celui relatif au mécanisme de diversification, *MaxFlips*, et ceux qui contrôlent l'application de la règle d'inférence sur la formule, *LimSUP* et *LimMR*.

MaxFlips Comme pour AdaptNovelty++, le meilleur pourcentage d'optima trouvés (figure 9.2a) est obtenu avec un nombre maximal de flips par relance faible (entre 5 et 25 fois la taille de la formule originale). Le temps médian pour trouver la première fois l'optimum (figure 9.2b) confirme cette observation et permet de l'affiner, en situant la meilleur valeur autour de dix. Enfin, il est intéressant de noter que *MaxFlips* n'a pas

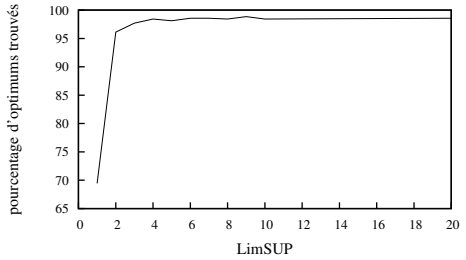


(a) Pourcentage d'optima trouvés (b) Temps médian pour atteindre l'optimum

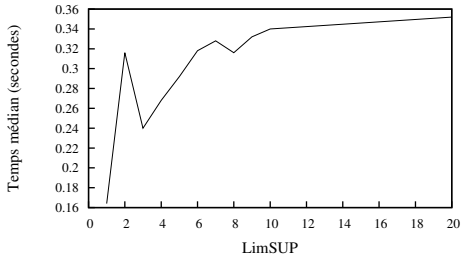
Figure 9.2 – Impact de la valeur de *MaxFlips* sur les performances IRAnovelty++.

d'impact sur la taille de la formule après transformation ni sur le nombre de clauses vides trouvées par l'algorithme (non représentés dans la figure 9.2).

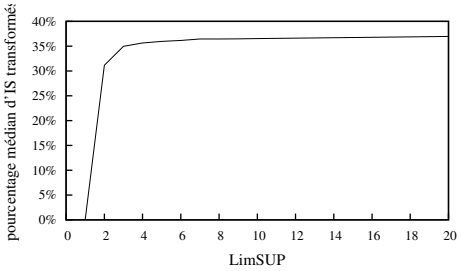
LimSUP Ce paramètre définit la taille maximale possible de la formule transformée, exprimée en un ratio de la taille de la formule originale. Quand la formule atteint ou dépasse cette taille, la propagation unitaire simulée n'est plus appliquée et la formule reste inchangée. Avec des valeurs inférieures à 1, la règle d'inférence n'est jamais appliquée et IRANOVELTY++ agit comme ADAPTNOVELTY++. En augmentant sa valeur, le pourcentage d'optima trouvés (figure 9.3a) augmente jusqu'à devenir stable autour de 98,5% avec des valeurs supérieures à 4. Le meilleur temps médian pour trouver l'optimum (figure 9.3b) est obtenu avec la valeur 8. Des valeurs plus grandes donnent des performances légèrement moins bonnes. Nous voyons deux raisons possibles expliquant ce comportement : (1) trop de temps est passé à transformer la formule et/ou (2) la taille plus importante de la formule transformée ralentit la recherche locale. La figure 9.3c montre le pourcentage de sous-ensembles inconsistants



(a) Pourcentage d'optima trouvés



(b) Temps médian pour atteindre l'optimum

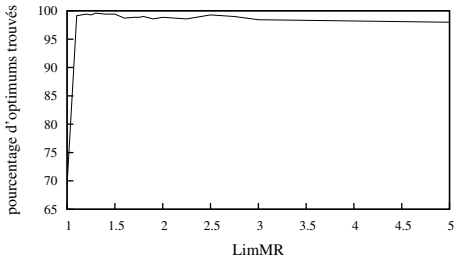


(c) Pourcentage médian de clauses vides générées

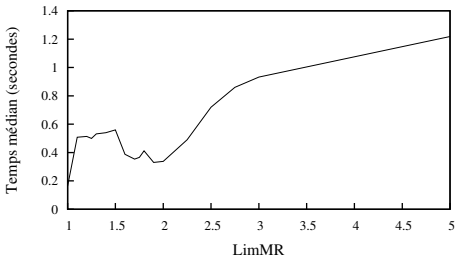
FIGURE 9.3 – Impact de la valeur de *LimSUP* sur les performances d'IRANOVELTY++.

LimMR Comme le précédent, ce paramètre contrôle l'application de la règle d'inférence sur la formule. Il limite la production de clauses résolvantes de tailles supérieures à un certain ratio de la plus petite des clauses en entrée. Pour un conflit donné, si la taille du résolvant est supérieure à *LimMR* fois celle de la plus petite clause du sous-ensemble inconsistant, alors la règle d'inférence n'est pas appliquée et le conflit est simplement ignoré. Comme nous l'avons expliqué précédemment, cette limite a deux objectifs : limiter la taille de la formule transformée et favoriser la production de clauses vides. Avec la valeur 1 la taille de la formule n'augmente quasiment pas et l'algorithme détecte près de 60% des clauses vides (figure 9.4c) mais il ne trouve l'optimum que sur moins de 70% des

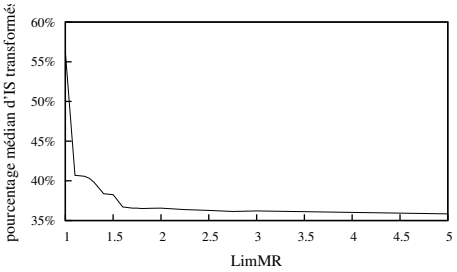
instances (figure 9.4a). Cela peut s'expliquer par le fait que *LimSUP* n'est jamais atteinte, donc SUP est appliquée durant toute la résolution et ralentit la partie recherche locale de l'algorithme. Avec des valeurs supérieures à 1, IRANOVELTY++ trouve l'optimum sur plus de 95% des instances et détecte environ 40% des clauses vides. Les meilleurs temps pour atteindre la première fois l'optimum (figure 9.4b) sont obtenus avec des valeurs de *LimMR* comprises entre 1,5 et 2.



(a) Pourcentage d'optima trouvés



(b) Temps médian pour atteindre l'optimum



(c) Pourcentage médian de clauses vides générées

FIGURE 9.4 – Impact de la valeur de *LimMR* sur les performances d'IRANOVELTY++.

9.3.2 Comparaison expérimentale

Nous comparons dans le reste de cette section les performances d'ADAPTNOVELTY++ et d'IRANOVELTY++ avec leurs paramètres optimisés (pour ADAPTNOVELTY++ *MaxFlips* = 1 et pour IRANOVELTY++ *MaxFlips* = 10, *LimSUP* = 5 and *LimMR* = 1.5). Nous incluons également dans cette comparaison trois solveurs de recherche locale pour Max-SAT : IROTS [SHS03], CCLS [LCW+14] et DIST [CLTS14]. Tous ont été régulièrement bien classés lors des *Max-SAT évaluation* récentes.

Les tests sont réalisés sur les instances non-partielles aléatoires et *crafted* du benchmark de la *Max-SAT Evaluation 2014*. Nous n'avons pas inclus d'instance Max-SAT partiel car ni ADAPTNOVELTY++ ni IRANOVELTY++ ne les gèrent correctement. Les algorithmes de recherche locale étant notoirement inefficaces sur les instances issues de problèmes industriels, nous avons également ignoré ces instances. Le temps et l'espace mémoire alloués aux solveur pour résoudre chaque instance sont limités à respectivement 300 secondes et 3,5 Go. Ces valeurs sont similaires à celles utilisées lors des *Max-SAT évaluations*. Pour limiter l'impact du caractère aléatoire de certains des paramètres des solveurs et évaluer leur robustesse, nous lançons chaque solveur dix fois sur chaque instance (10 *runs*).

Les résultats sont présentés dans les tables 9.1 et 9.2. Pour chaque solveur, les colonnes $\%O$ et T donnent respectivement le pourcentage de *runs* où l'optimum a été trouvé et le temps médian nécessaire pour l'atteindre. Pour chaque classe d'instances, le meilleur résultat est présenté en gras et le meilleur résultat entre AdaptNovelty++ et IRAnovelty++ est souligné.

AdaptNovelty++ vs. IRAnovelty++ On peut tout d'abord remarquer que la variante utilisant les règles d'inférence, IRAnovelty++, trouve l'optimum sur environ 83% des runs réalisés tandis qu'AdaptNovelty++ le trouve sur environ 56% d'entre eux. Le gain est particulièrement marqué sur les instances max-2-sat (e.g. ms/crafted/bipartite, ms/random/max2sat ou ms/random/min2sat) où le taux de solutions optimales atteintes progresse sensiblement (respectivement +62,4, +65,8 et +92,5 points de pourcentage). À l'inverse, sur les instances max-k-sat avec $k > 2$ (e.g. ms/randomn/highgirt ou ms/random/max3sat) on peut observer que le temps moyen nécessaire pour atteindre les solutions optimales augmente et dans certains cas le taux de solutions optimales atteintes est réduit.

Table 9.1 – Comparaison des performances d'AdaptNovelty++ et d'IRAnovelty++. Pour chaque solveur, les colonnes $\%O$ et T donnent respectivement le pourcentage de *runs* où une solution optimale a été trouvée et le temps médian nécessaire pour l'atteindre.

Classes d'instances			Anovelty++		IRAnovelty++	
			%O	T	%O	T
ms	crafted	bipartite	0,4%	213,29	**62,8%**	**149,35**
		maxcut	91%	6,4	**96,4%**	**0,37**
		set-covering	8%	274,85	**20%**	**41,72**
	random	highgirth	**94,4%**	**15,41**	86,3%	99,58
		max2sat	0,2%	295,18	**66%**	**166,01**
		max3sat	**100%**	**0,41**	100%	3,04
		min2sat	0,2%	326,67	**92,7%**	**5,1**
wms	crafted	frb	**29,4%**	**52,91**	29,4%	84,37
		ramsey	**68%**	**4,08**	61,3%	24,66
		wmaxcut	95,4%	0,22	**96,3%**	**0,42**
	rand	wmax2sat	88,3%	9,08	**95%**	**24,49**
		wmax3sat	**97,5%**	**0,11**	97,5%	0,98
		Ensemble	55,5%	1,95	**83%**	**19,77**

Si l'on considère le temps médian nécessaire pour trouver une solution optimale, on peut remarquer qu'AdaptNovelty++ est généralement plus rapide qu'IRAnovelty++. Cela peut s'expliquer par deux faits : (1) l'application de la règle d'inférence est coûteuse en temps, particulièrement au début de la recherche (sur la plupart des instances, *LimSUP* est atteinte dans les toutes premières secondes de la résolution) et donc peut retarder la découverte de l'optimum et (2) la taille plus grande de la formule transformée ralentit la recherche locale et donc la découverte de l'optimum.

Les résultats décevants d'IRAnovelty++ sur les instances max-k-sat avec $k > 2$ peut s'expliquer comme suit. Nous supposons que la génération de clauses vides permet de simplifier la résolution des instances. À part dans certains cas particuliers (quand plusieurs clauses de l'arbre de réfutation contiennent des littéraux communs), les clauses de compensation et les résolvants produits par la règle d'inférence sont de tailles égales

(avec des clauses originales de tailles 2) ou supérieures (avec des clauses originales de tailles > 2) à celles des clauses originales. Sur les instances Max-2-SAT, le cas particulier présenté ci-dessus est suffisamment fréquent pour permettre la production de clauses unitaires et vides. Ce n'est visiblement pas le cas sur les instances avec des clauses de plus grandes tailles et en conséquence IRAnovelty++ passe du temps à transformer la formule, augmentant ainsi sa taille, sans pour autant générer de clauses vides et donc sans simplifier la formule.

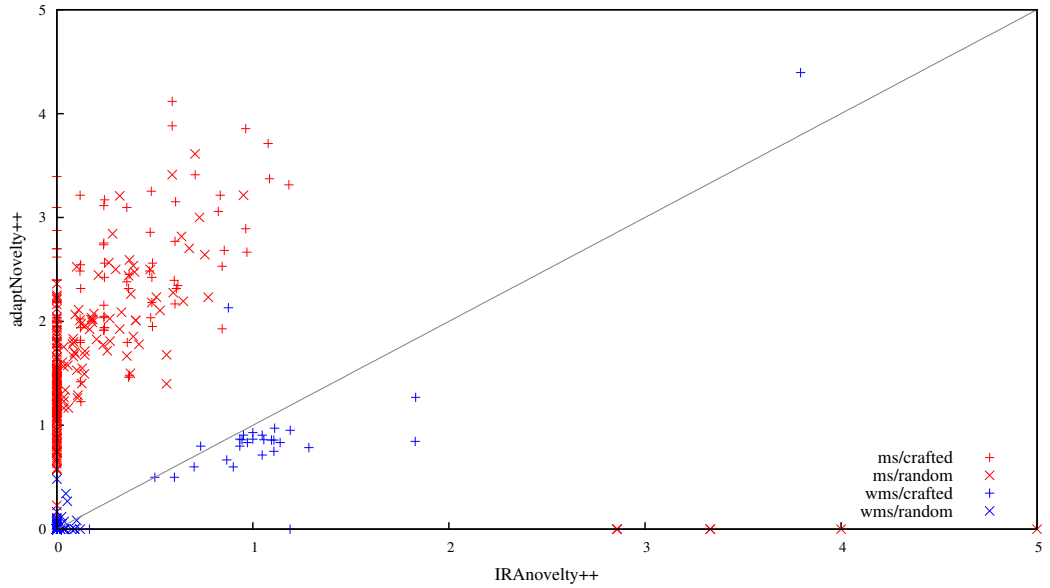
Il est intéressant de noter que sur certaines instances IRAnovelty++ trouve et transforme tous les sous-ensembles inconsistants de la formule. Dans cette situation, la somme des poids des clauses vides est l'optimum de l'instance et l'ensemble des autres clauses de la formule est satisfiable. IRAnovelty++ peut donc parfois prouver l'optimalité comme le font les solveurs Max-SAT complets. Cependant, ce cas est rare et ne concerne qu'une poignée d'instances ayant de petites valeurs d'optima (c'est-à-dire dont le nombre minimum de clauses falsifiées est faible).

La figure 9.5 compare instance par instance la distance à l'optimum (exprimée en pourcentage de sa valeur) et le temps moyen nécessaire pour l'atteindre. On peut voir que la distance à l'optimum (figure 9.5a) d'IRAnovelty++ est généralement plus faible que celle d'AdaptNovelty++, excepté sur la classe d'instances wms/crafted. En revanche, sur les instances pour lesquelles les deux solveurs trouvent une solution optimale, le temps nécessaire à la découverte d'une telle solution (figure 9.5b) est sensiblement plus faible pour AdaptNovelty++ que pour IRAnovelty++. Cela confirme les observations et interprétations faites précédemment.

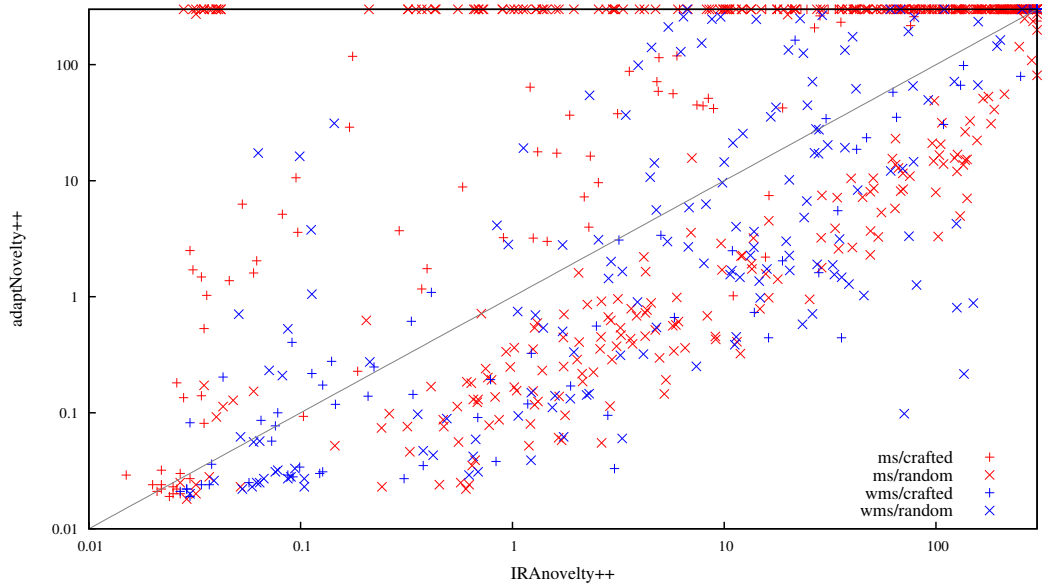
IRAnovelty++ vs. état de l'art On peut noter qu'AdaptNovelty++ et IRAnovelty++ sont sensiblement moins performants que les autres solveurs de recherche locale considérés, que ce soit en termes de taux d'optima atteints ou de temps nécessaire pour les atteindre. Cela vient confirmer les observations faites par Zhang *et al.* [ZRL03] ou Cai *et al.* [CLTS14] selon lesquels les solveurs de type WalkSat (i.e. qui choisissent les va-

Table 9.2 – Comparaison des performances d'IRAnovelty++ avec des solveurs de recherche locale de l'état de l'art. Pour chaque solveur, les colonnes $\%O$ et T donnent respectivement le pourcentage de *runs* où une solution optimale a été trouvée et le temps médian nécessaire pour l'atteindre.

Classes d'instances			IRAnovelty++		irots		ccls		Dist	
			%O	T	%O	T	%O	T	%O	T
ms	crafted	bipartite	62,8%	149,35	99,8%	0,06	**99,8%**	**0,05**	99,8%	0,06
		maxcut	96,4%	0,37	**100%**	**0,04**	**100%**	**0,04**	96,4%	0,04
		set-covering	20%	41,72	20%	83,44	**73%**	**0,63**	50%	6,89
	random	highgirth	86,3%	99,58	98,3%	1,13	**98,8%**	**1,71**	29,1%	57,6
		max2sat	66%	166,01	**100%**	**0,04**	100%	0,05	100%	0,05
		max3sat	100%	3,04	**100%**	**0,04**	**100%**	**0,04**	**100%**	**0,04**
		min2sat	92,7%	5,1	**100%**	**0,04**	**100%**	**0,04**	100%	0,05
wms	crafted	frb	29,4%	84,37	43,2%	0,16	**97,9%**	**0,2**	89,7%	0,67
		ramsey	61,3%	24,66	80%	1,05	**80,7%**	**1,46**	76%	0,69
		wmaxcut	96,3%	0,42	**97,9%**	**0,05**	**97,9%**	**0,05**	93,3%	0,05
	rand	wmax2sat	95%	24,49	**100%**	**0,04**	100%	0,05	100%	0,05
		wmax3sat	97,5%	0,98	**100%**	**0,04**	100%	0,05	100%	0,06
Ensemble			83%	19,77	95,9%	0,04	**99%**	**0,05**	90,6%	0,05



(a) Distance moyenne à l'optimum en pourcentage.



(b) Temps moyen pour atteindre l'optimum en seconde (échelles logarithmiques).

FIGURE 9.5 – Comparaison instance par instance d'ADAPTNOVELTY++ et d'IRANOVELTY++.

riables à flipper dans une clause falsifiée sélectionné aléatoirement) sont peu performants sur les instances fortement contraintes.

9.4 Conclusion

Nous avons présenté dans ce chapitre un nouvel algorithme, IRANOVELTY++, qui intègre des règles d'inférence dans un solveur de recherche locale classique pour résoudre le problème Max-SAT. Les tests que nous avons effectués donnent des informations utiles sur le comportement de notre algorithme et prouvent sa robustesse sur un grand nombre d'instances. Les résultats expérimentaux que nous avons obtenus montrent que les performances sont nettement améliorées par rapport à celles de l'algorithme original, ce qui confirme l'intérêt de notre approche. Ils montrent également que les performances d'ADAPTNOVELTY++ et d'IRANOVELTY++ sont inférieures à celle des solveurs de recherche locale de l'état de l'art.

Dans le futur, nous étendrons notre algorithme au problème partial Max-SAT et nous étudierons plus en détail sont comportement sur les instances Max-3-SAT, avec en perspective l'objectif de permettre la génération de clauses vides sur ces instances, par exemple en guidant l'application de la règle d'inférence par des heuristiques dédiées. Nous essaierons d'étendre les résultats obtenues ici en appliquant la règle d'inférence à d'autres algorithmes de recherche locale, et en particulier aux plus performants sur les instances du problème Max-SAT.

Enfin, les travaux récents sur le problème Max-SAT ont majoritairement porté sur sa résolution par des méthodes complètes, tandis que moins d'intérêt a été porté sur les méthodes incomplètes. Notre travail à aussi pour but de participer au développement de ces approches.

Conclusion générale

Nous nous sommes intéressés dans cette thèse au problème de satisfiabilité maximum Max-SAT. Parmi les méthodes complètes élaborées pour résoudre Max-SAT, les solveurs de type séparation et évaluation ont montrés leur efficacité principalement sur les instances aléatoires. Cependant, ils présentent certains inconvénients : les traitements qu'ils appliquent sont très redondants (et donc coûteux en temps de calcul) et ils sont incapables de prendre en compte la structure des instances pour faciliter la résolution.

Dans ce contexte, une des pistes prometteuses pour améliorer les solveurs de type séparation et évaluation pour Max-SAT passe par l'apprentissage. Il pourrait permettre de réduire la redondance dans les calculs effectués lors du parcours de l'arbre de recherche et de mieux prendre en compte la structure des instances. Nous avons donc étudié les méthodes permettant de réaliser une forme d'apprentissage : la combinaison de la propagation unitaire et de la max-résolution. Nous avons cherché à mieux exploiter leur potentiel et à mieux comprendre leur fonctionnement. Nos contributions, qui ont toutes fait l'objet de publications dans des conférences internationales [AH12, AH14d, AH14e, AH14a, AH14c, AH15c] et nationales [AH13, AH14b, AH15a, AH15b, AH15d], sont les suivantes.

Nous avons réduit le temps consacré au calcul de la borne inférieure en limitant la redondance dans l'application de la propagation unitaire (par le schéma de propagation MPS, cf. chapitre 3) et dans la détection et le traitement des sous-ensembles inconsistants (par l'inclusion des UCS dans le schéma d'apprentissage, cf. chapitre 6). La limitation du nombre et de la taille des clauses de compensation produites par les transformations par max-résolution (par l'heuristique SIR, cf. chapitre 4) contribue également à cette réduction.

Nous avons amélioré la qualité de l'estimation de la borne inférieure en réduisant la taille des clauses de compensation (par l'heuristique SIR), qui sont donc plus à même d'être utilisées par la propagation unitaire, et en appliquant les transformations par max-résolution localement à chaque nœud (cf. chapitre 5) ce qui permet de laisser plus de clauses disponibles pour appliquer la propagation unitaire et donc de détecter davantage de sous-ensembles inconsistants.

Nous avons limité un des principaux inconvénients connus des transformations par max-résolution : l'augmentation de la taille de la formule (toujours par l'heuristique SIR). Nous avons également caractérisé l'impact de ces transformations sur la propagation unitaire (par la notion d'UP-résilience, cf. chapitre 7), ce qui contribue à expliquer d'un point de vue théorique l'efficacité des schémas d'apprentissage existants.

Le solveur AHMAXSAT développé durant cette thèse s'est distingué lors de la *Max-SAT Evaluation 2014* où il a été classé premier dans trois des neufs catégories d'instances et second dans une quatrième. Ces résultats sont confirmés par notre propre étude expéri-

mentale (cf. chapitre 8).

Enfin, nous avons également donné un exemple d'intégration des règles de la propagation unitaire et de la recherche locale dans un algorithme de recherche locale. Les résultats obtenus montrent qu'elles permettent d'améliorer l'efficacité de la recherche locale, même si ces résultats méritent d'être étendus à des algorithmes de recherche locale plus compétitifs.

Les contributions que nous avons apportées permettent d'envisager plusieurs perspectives. En effet, nous avons fourni des éléments permettant de mieux comprendre les effets des transformations par max-résolution et de contrôler certains de leurs inconvénients. L'utilisation de cette règle comme une méthode d'apprentissage généralisée pourrait être mieux appréhendée. On peut ainsi envisager d'étendre l'apprentissage à la fois vers le bas de l'arbre de recherche comme le font les solveurs existants mais également vers le haut de l'arbre. Dans le futur, nous chercherons tout d'abord à compléter les schémas d'apprentissage existants, soit en ajoutant de nouveaux motifs aux schémas actuels soit en élaborant une méthode efficace pour déterminer le pourcentage d'UP-résilience des transformations tout au long de la recherche. Dans un second temps, nous envisagerons la conservation des transformations apprises vers le haut de l'arbre de recherche. Une méthode possible serait de recalculer leur pourcentage d'UP-résilience à chaque niveau de l'arbre de recherche et de conserver les modifications tant que ce pourcentage reste acceptable. Enfin, nous avons vu que l'ordre d'application des étapes de max-résolution a un impact sur l'UP-résilience des transformations. Nous étudierons ce phénomène en cherchant à concilier la minimisation du nombre et de la taille des sous-ensembles inconsistants et la maximisation du pourcentage d'UP-résilience. Nous élargirons cette étude à d'autres composants des solveurs, pour chercher à influer sur l'UP-résilience des transformations et par là même sur l'apprentissage effectué par les solveurs.

D'autres méthodes d'apprentissage sont envisageables. Par exemple, lorsqu'à un nœud de l'arbre de recherche la borne supérieure est inférieure ou égale à la borne inférieure on sait que l'interprétation partielle courante (ou un sous-ensemble de cette interprétation) ne peut pas améliorer la meilleure solution déjà rencontrée. On peut donc ajouter une clause dure à la formule pour empêcher de revenir à cette interprétation. Cette méthode d'apprentissage serait assez similaire à celle appliquée dans les solveurs SAT de type CDCL et on pourrait probablement lui adjoindre les techniques développées dans ce cadre : retour-arrières non chronologiques, *first unique implication point* et clause assertive et des heuristiques de branchement basées sur les apparitions des variables dans les conflits.

Une autre piste que nous étudierons concerne la cohabitation entre les techniques issues des solveurs SAT complets et celles développées dans le cadre des solveurs de type séparation et évaluation pour Max-SAT. Nous projetons d'intégrer à AHMAXSAT certaines d'entre elles et d'étudier leurs interactions avec les autres composants du solveur.

Enfin, nous avons vu que les transformations par max-résolution entraînaient une reformulation des instances, et que les règles d'inférence utilisées lors de la résolution (dans notre cas la propagation unitaire) étaient affectées par ces reformulations. Ce phénomène pourrait apparaître dans d'autres problèmes d'optimisation, comme le problème de satisfaction de contraintes valuées. Nous étudierons cet aspect en cherchant à étendre l'apprentissage pour d'autres problèmes d'optimisation et à identifier des points communs entre ces phénomènes.

Bibliographie

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity : A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[ABGL12] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 86–101, 2012.

[ABH+08] Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In Hans Kleine Büning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT 2008)*, volume 4996 of *Lecture Notes in Computer Science*, pages 21–27. Springer Berlin Heidelberg, 2008.

[ABL09a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On solving maxsat through sat. In *Artificial Intelligence Research and Development, Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence, CCIA 2009, October 21-23, 2009, Vilar Rural de Cardona (El Bages), Cardona, Spain*, pages 284–292, 2009.

[ABL09b] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer Berlin Heidelberg, 2009.

[ABL10] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *AAAI*, pages 3–8, 2010.

[ABL13] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artificial Intelligence*, 196(0) :77 – 105, 2013.

[Ach09] Tobias Achterberg. SCIP : solving constraint integer programs. *Mathematical Programming Computation*, 1(1) :1–41, 2009.

[AG13] Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial maxsat with ilp. In Carla P. Gomes and Meinolf Sellmann, editors, *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*, volume 7874 of *Lecture Notes in Computer Science*, pages 403–409. Springer, 2013.

[AH12] André Abramé and Djamal Habet. Inference rules in local search for max-sat. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, pages 207–214. IEEE Computer Society, 2012.

[AH13] André Abramé and Djamal Habet. Règles d'inférence et recherche locale pour max-sat et max-sat valué. In *Actes des 9ème Journées Francophone de la Programmation par Contrainte (JFPC 2013)*, 2013.

[AH14a] André Abramé and Djamal Habet. Efficient application of max-sat resolution on inconsistent subsets. In Barry O'Sullivan, editor, *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*, volume 8656 of *Lecture Notes in Computer Science*, pages 92–107. Springer International Publishing, 2014.

[AH14b] André Abramé and Djamal Habet. La max-résolution locale dans les solveurs séparation & evaluation pour max-sat. In *Actes des 10èmes Journées Francophones de la Programation par Contrainte (JFPC 2014)*, 2014.

[AH14c] André Abramé and Djamal Habet. Local max-resolution in branch and bound solvers for max-sat. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014)*, pages 336–343. IEEE Computer Society, 2014.

[AH14d] André Abramé and Djamal Habet. Maintaining and handling all unit propagation reasons in exact max-sat solvers. In Stefan Edelkamp and Roman Barták, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS 2014)*, pages 2–9. AAAI Press, 2014.

[AH14e] André Abramé and Djamal Habet. On the extension of learning for max-sat. In Ulle Endriss and João Leite, editors, *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*, volume 241 of *Frontiers in Artificial Intelligence and Applications*, pages 1–10. IOS Press, 2014.

[AH15a] André Abramé and Djamal Habet. Application efficace de la résolution pour max-sat sur les ensembles inconsistants. In *Actes des 11èmes Journées Francophones de Programmation par Contrainte (JFPC 2015)*, 2015.

[AH15b] André Abramé and Djamal Habet. Maintenir et traiter toutes les sources de propagation unitaire dans les solveurs exacts pour max-sat. In *Actes des 11èmes Journées Francophones de Programmation par Contrainte (JFPC 2015)*, 2015.

[AH15c] André Abramé and Djamal Habet. On the resiliency of unit propagation to max-resolution. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

[AH15d] André Abramé and Djamal Habet. Vers l'extension de l'apprentissage pour max-sat. In *Actes des 11èmes Journées Francophones de Programmation par Contrainte (JFPC 2015)*, 2015.

[Ake78] S.B. Akers. Binary decision diagrams. *Computers, IEEE Transactions on*, C-27(6) :509–516, June 1978.

[ALMS09] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. Learning in local search. In *21st IEEE International Conference on Tools with Artificial Intelligence - ICTAI 2009*, pages 417–424. IEEE Computer Society, 2009.

[ALMS10] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. Boosting local search thanks to cdcl. In Christian Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *LNCS*, pages 474–488. Springer Berlin / Heidelberg, 2010.

[AMP03] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved branch and bound algorithms for max-sat. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.

[AMP04] Teresa Alsinet, Felip Manyà, and Jordi Planes. A max-sat solver with lazy data structures. In Christian Lemaître, Carlos Reyes, and Jesùs Gonzàlez, editors, *Proceedings of the 9th Ibero-American on Artificial Intelligence (IBERAMIA 2004)*, volume 3315 of *Lecture Notes in Computer Science*, pages 334–342. Springer Berlin / Heidelberg, 2004.

[AMS14] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. Maxsat by improved instance-specific algorithm configuration. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2594–2600. AAAI Press, 2014.

[Anj06] Miguel F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(1) :1–47, 2006.

[ASM85] Patrizia Asirelli, Michelle De Santis, and Maurizio Martelli. Integrity constraints in logic databases. *The Journal of Logic Programming*, 2(3) :221 – 232, 1985.

[BB92] M. Buro and H. Kleine Büning. Report on a sat competition. Technical report, University of Paderborn, 1992.

[BF98] Brian Borchers and Judith Furman. A two-phase exact algorithm for max-sat and weighted max-sat problems. *Journal of Combinatorial Optimization*, 2 :299–306, 1998.

[BF10] Adrian Balint and Andreas Fröhlich. Improving stochastic local search for sat with a new probability distribution. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 10–15. Springer Berlin Heidelberg, 2010.

[BGS99] Laure Brisoux, Éric Grégoire, and Lakhdar Saïs. Improving backtrack search for sat by means of redundancy. In ZbigniewW. Raś and Andrzej Skowron, editors, *Foundations of Intelligent Systems*, volume 1609 of *Lecture Notes in Computer Science*, pages 301–309. Springer Berlin Heidelberg, 1999.

[BHvMW09] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BLM07] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artificial Intelligence*, 171(8-9) :606–618, 2007.

[BNV+13] Howard E. Bond, Edmund P. Nelan, Don A. VandenBerg, Gail H. Schaefer, and Dianne Harmer. Hd 140283 : A star in the solar neighborhood that formed shortly after the big bang. *The Astrophysical Journal Letters*, 765(1) :L12, 2013.

[Boo54] George Boole. *An Investigation of the Laws of Thought.* Walton and Maberly, 1854.

[BP10] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3) :59–64, 2010.

[BR99] Nikhil Bansal and Venkatesh Raman. Upper bounds for maxsat : Further improved. In *Algorithms and Computation*, volume 1741 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin / Heidelberg, 1999.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3) :293–318, 1992.

[Cer85] V. Cerny. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1) :41–51, January 1985.

[CIKM97] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial maxsat. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 97)*, pages 263–268. AAAI Press / The MIT Press, 1997.

[CLTS14] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial maxsat. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 2623–2629. AAAI Press, 2014.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM Symposium on Theory of Computing - STOC'71*, pages 151–158. ACM, 1971.

[CS11] Shaowei Cai and Kaile Su. Local search with configuration checking for sat. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 59–66, Nov 2011.

[CS12] Shaowei Cai and Kaile Su. Configuration checking with aspiration in local search for sat. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence - AAAI 2012.* AAAI Press, 2012.

[CS13] Shaowei Cai and Kaile Su. Comprehensive score : Towards efficient local search for sat with long clauses. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[CSL13] Shaowei Cai, Kaile Su, and Chuan Luo. Improving walksat for random k-satisfiability problem with k $>$ 3. In *Proceedings of the Twenty-Seventh*

AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA., 2013.

[CSS11] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9–10) :1672–1696, 2011.

[DB11] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Lee, editor, *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP 2011)*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer Berlin Heidelberg, 2011.

[DB13a] Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[DB13b] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up maxsat solving. In Christian Schulte, editor, *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[DCB10] Jessica Davies, Jeremy Cho, and Fahiem Bacchus. Using learnt clauses in maxsat. In David Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 176–190. Springer Berlin / Heidelberg, 2010.

[DD04] Gilles Dequen and Olivier Dubois. kcnfs : An efficient solver for random k-sat formulae. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 486–501. Springer Berlin Heidelberg, 2004.

[DDDL07] Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu-Min Li. On inconsistent clause-subsets for max-sat solving. In Christian Bessière, editor, *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *LNCS*, pages 225–240. Springer Berlin / Heidelberg, 2007.

[DG12] D. D'Almeida and E. Gregoire. Model-based diagnosis with default information implemented through max-sat technology. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 33–36, 2012.

[dGHZL05] Simon de Givry, Federico Heras, Matthias Zytnicki, and Javier Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted csps. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 84–89. Professional Book Center, 2005.

[dGLMS03] Simon de Givry, Javier Larrosa, Pedro Meseguer, and Thomas Schiex. Solving max-sat as weighted csp. In Francesca Rossi, editor, *Principles and*

Practice of Constraint Programming – CP 2003, volume 2833 of *Lecture Notes in Computer Science*, pages 363–376. Springer Berlin Heidelberg, 2003.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, July 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3) :201–215, 1960.

[ES03] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[FM06] Zhaohui Fu and Sharad Malik. On solving the partial max-sat problem. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin Heidelberg, 2006.

[Fre95] J. W. Freeman. *Improvements To Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[FW92] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3) :21–70, 1992.

[Gal77] Zvi Galil. On the complexity of regular resolution and the davis-putnam procedure. *Theoretical Computer Science*, 4(1) :23 – 46, 1977.

[GH11] Oliver Gableske and Marijn Heule. Eagleup : Solving random 3-sat using sls with unit propagation. In Karem Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695 of *LNCS*, pages 367–368. Springer Berlin / Heidelberg, 2011.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GKK+11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco : The potsdam answer set solving collection. *AI Communications*, 24(2) :107–124, 2011.

[GKKS12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[GL12] João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 941–956. Springer Berlin Heidelberg, 2012.

[Glo89] Fred Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3) :190–206, Summer 1989.

[Gol79] Allen T Goldberg. On the complexity of the satisfiability problem. Technical report, New York University, 1979.

[GP95] Jun Gu and R. Puri. Asynchronous circuit synthesis with boolean satisfiability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(8) :961–973, Aug 1995.

[GvHL06] Carla Gomes, Willem-Jan van Hoeve, and Lucian Leahu. The power of semidefinite programming relaxations for max-sat. In J. Christopher Beck and Barbara Smith, editors, *Proceedings of the 3rd Interantional Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2006)*, volume 3990 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin / Heidelberg, 2006.

[GW94a] Michel X. Goemans and David P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2sat. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC 1994)*, pages 422–431. ACM, 1994.

[GW94b] Michel X. Goemans and David P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7(4) :656–666, 1994.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6) :1115–1145, 1995.

[Hås97] Johan Håstad. Some optimal inapproximability results. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC 1997)*, pages 1–10. ACM, 1997.

[Hås01] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4) :798–859, 2001.

[HJ90] Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44 :279–303, 1990.

[HK05] Edward A. Hirsch and Arist Kojevnikov. Unitwalk : A new sat solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43 :91–111, 2005.

[HL06] Federico Heras and Javier Larrosa. New inference rules for efficient max-sat solving. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 68–73. AAAI Press, 2006.

[HLO07] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat : A new weighted max-sat solver. In João Marques-Silva and Karem Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, volume 4501 of *Lecture Notes in Computer Science*, pages 41–55. Springer Berlin / Heidelberg, 2007.

[HLO08] Federico Heras, Javier Larrosa, and Albert Oliveras. Minimaxsat : An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31 :1–32, 2008.

[HMMS11] Federico Heras, António Morgado, and João Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence - AAAI 2011*. AAAI Press, 2011.

[HMS11] Federico Heras and João Marques-Silva. Read-once resolution for unsatisfiability-based max-sat algorithms. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 572–577. AAAI Press, 2011.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Hol92] J. H. Holland. *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Application to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[Hoo99] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 661–666, 1999.

[Hoo02] H. H. Hoos. An adaptive noise mechanism for walksat. In *Proceedings of the 18th National Conference on Artificial Intelligence - AAAI'02*, pages 655–660, 2002.

[HS04] H. Hoos and T. Stutzle. *Stochastic Local Search : Foundations and Applications*. Morgan Kaufmann Publishers Inc., 2004.

[HT12] Djamal Habet and Donia Toumi. Local search based on conflict analysis for the satisfiability problem. In *Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI 2012)*, pages 892–897. IEEE Computer Society, 2012.

[HTH02] Frank Hutter, DaveA.D. Tompkins, and HolgerH. Hoos. Scaling and probabilistic smoothing : Efficient dynamic local search for sat. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 of *Lecture Notes in Computer Science*, pages 233–248. Springer Berlin Heidelberg, 2002.

[HV95] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3) :359–383, 1995.

[JKS95] Yuejun Jiang, Henry Kautz, and Bart Selman. Solving problems with hard and soft constraints using a stochastic algorithm for max-sat. In *Proceedings of the 1st International Joint Workshop on Artificial Intelligence and Operations Research*, 1995.

[JMB97] Steve Joy, John Mitchell, and Brian Borchers. A branch and cut algorithm for max-sat and weighted max-sat. In Dingzhu Du, Jun Gu, and Panos M. Pardalos, editors, *Satisfiability Problem : Theory and Applications, volume 35 of DIMACS series on Discrete Mathematics and Theoretical Computer Science*, pages 519–536. American Mathematical Society, 1997.

[Joh73] David S. Johnson. Approximation algorithms for combinatorial problems. In Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 38–49. ACM, 1973.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3) :256–278, 1974.

[JW90] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1 :167–187, 1990.

[KGV83] S. Kirkpatrick, D. Gelatt, and M. P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598) :671–680, 1983.

[Küg10] Adrian Kügel. Improved exact solver for the weighted max-sat problem. In Daniel Le Berre, editor, *Proceedings of the 1st Pragmatics of SAT Workshop (POS 2010)*, volume 8 of *EasyChair Proceedings in Computing*, pages 15–27. EasyChair, 2010.

[Küg12] Adrian Kügel. Natural max-sat encoding of min-sat. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 431–436. Springer Berlin Heidelberg, 2012.

[KZ97] Howard J. Karloff and Uri Zwick. A 7/8-approximation algorithm for MAX 3sat ? In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS 1997)*, pages 406–415. IEEE Computer Society, 1997.

[KZFH12] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat : A partial max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2) :95–100, 2012.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In G. Smolka, editor, *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 341–355, Linz, Austria, November 1997. Springer.

[LCW+14] C. Luo, S. Cai, W. Wu, Z. Jie, and K. Su. Ccls : An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, PP(99) :1–1, 2014.

[LD60] A H Land and A G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3) :497–520, 1960.

[Lev73] Leonid A Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3) :115–116, 1973.

[LH05a] Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 193–198. Professional Book Center, 2005.

[LH05b] Chu Min Li and Wen Huang. Diversification and determinism in local search for satisfiability. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT'05*, volume 3569 of *LNCS*, pages 158–172. Springer Berlin / Heidelberg, 2005.

[LH12] Zhipeng Lü and Jin-Kao Hao. Adaptive memory-based local search for max-sat. *Applied Soft Computing*, 12(8) :2063–2071, 2012.

[LHdG08] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artificial Intelligence*, 172(2-3) :204–233, 2008.

[LL12] Chu Min Li and Yu Li. Satisfying versus falsifying in local search for satisfiability. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 477–478. Springer Berlin Heidelberg, 2012.

[LMMP09] Chu Min Li, Felip Manyà, Nouredine Mohamedou, and Jordi Planes. Exploiting cycle structures in max-sat. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *Lecture Notes in Computer Science*, pages 467–480. Springer Berlin / Heidelberg, 2009.

[LMMP10] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in maxsat. *Constraints*, 15(4) :456–484, 2010.

[LMP05] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In Peter van Beek, editor, *Proceedings of the 11th Interantional Conference on Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 403–414. Springer Berlin / Heidelberg, 2005.

[LMP06] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*, pages 86–91. AAAI Press, 2006.

[LMP07] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for max-sat. *Journal of Artificial Intelligence Research*, 30 :321–359, 2007.

[LMQZ10] Chu Min Li, Felip Manyà, Zhe Quan, and Zhu Zhu. Exact minsat solving. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing – SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 363–368. Springer Berlin Heidelberg, 2010.

[LMS02] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, 2002.

[LS03] Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted csp. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 239–244. Morgan Kaufmann, 2003.

[LSB04] Xiao Li, Matthias Stallmann, and Franc Brglez. A local search sat solver using an effective switching strategy and an efficient unit propagation. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 276–277. Springer Berlin / Heidelberg, 2004.

[LSH05] Frédéric Lardeux, Frédéric Saubion, and Jin-Kao Hao. Three truth values for the sat and max-sat problems. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI'05*, pages 187–192. Professional Book Center, 2005.

[LWL12] Chu Min Li, Wanxia Wei, and Yu Li. Exploiting historical relationships of clauses and variables in local search for satisfiability. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, volume 7317 of *Lecture Notes in Computer Science*, pages 479–480. Springer Berlin Heidelberg, 2012.

[MDM14] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In Barry O'Sullivan, editor, *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[MHM12] António Morgado, Federico Heras, and João Marques-Silva. Improvements to core-guided binary search for maxsat. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, pages 284–297, 2012.

[MM08] João Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, pages 225–230, 2008.

[MML10] Vasco M. Manquinho, Ruben Martins, and Inês Lynce. Improving unsatisfiability-based algorithms for boolean optimization. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 181–193, 2010.

[MML12] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. On partitioning for maximum satisfiability. In Luc De Raedt, Christian Bessière, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 913–914. IOS Press, 2012.

[MML13] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Community-based partitioning for maxsat solving. In Allen Van Gelder Matti Järvisalo, editor, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 of *Lecture Notes in Computer Science*, pages 182–191. Springer Berlin / Heidelberg, 2013.

[MML14] Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo : A modular maxsat solver,. In Carsten Sinz and Uwe Egly, editors, *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014)*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer International Publishing, 2014.

[MMSP09] Vasco Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer Berlin Heidelberg, 2009.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC 2001)*, pages 530–535. ACM, 2001.

[Mor93] Paul Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence - AAAI'93*, pages 40–45. AAAI Press, 1993.

[MP07] João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, 2007.

[MSG97] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Tabu search for sat. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 281–285, 1997.

[MSK97] David A. McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference - AAAI'97/IAAI'97*, pages 321–326. AAAI Press / The MIT Press, 1997.

[MSP08] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe (DATE 2008)*, pages 408–413. IEEE, 2008.

[MSS99] João Marques-Silva and Karem A. Sakallah. Grasp : A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, August 1999.

[MT00] Patrick Mills and Edward Tsang. Guided local search for solving sat and weighted max-sat problems. *Journal of Automated Reasoning*, 24 :205–223, 2000.

[MZK+99] Remi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic /'phase transitions/'. *Nature*, 400(6740) :133–137, July 1999.

[NB14] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAA1 2014)*, pages 2717–2723. AAAI Press, 2014.

[NR00] Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36(1) :63 – 88, 2000.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[PBTN12] A. Prugel-Bennett and M.-H. Tayarani-Najaran. Maximum satisfiability : Anatomy of the fitness landscape for a hard combinatorial optimization problem. *Evolutionary Computation, IEEE Transactions on*, 16(3) :319–338, 2012.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. Clone : Solving weighted max-sat in a reduced search space. In Mehmet Orgun and John Thornton, editors, *Advances in Artificial Intelligence - AI 2007*, volume 4830 of *LNCS*, pages 223–233. Springer Berlin / Heidelberg, 2007.

[POT] Potsdam answer set solving collection. `http://potassco.sourceforge.net/`.

[PPC+08] Knot Pipatsrisawat, Akop Palyan, Mark Chavira, Arthur Choi, and Adnan Darwiche. Solving weighted max-sat problems in a reduced search space : A performance analysis. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4) :191–217, 2008.

[Pre93] Daniele. Pretolani. Efficiency and stability of hypergraph sat algorithms. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability : Proceedings of the Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 479–498. American Mathematical Society, 1993.

[PTGS08] Duc Nghia Pham, John Thornton, Charles Gretton, and Abdul Sattar. Combining adaptive and dynamic local search for satisfiability. *JSAT*, 4(2-4) :149–172, 2008.

[Qui50] William V. Quine. *Methods of Logic*. Henry Holt, 1950.

[RM08] O. Roussel and V. Manquinho. *Handbook of satisfiability*, chapter Pseudo-Boolean and cardinality constraints, pages 695–734. IOS Press, 2008.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, 1965.

[Sai08] Lakhdar Sais, editor. *Problème SAT : Progrès et Défis*. Hermes Science Publications, 2008.

[SBS05] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3) :389–402, 2005.

[SE08] N. Sorensson and N. Een. Minisat 2.1 and minisat++ 1.0 — sat race 2008 editions. Technical report, 2008.

[SFV95] Thomas Schiex, Helene Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems : hard and easy problems. In *Proceedings of the 14th international joint conference on Artificial intelligence*, volume 1, pages 631–637, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[SHS03] Kevin Smyth, Holger Hoos, and Thomas Stützle. Iterated robust tabu search for max-sat. In Yang Xiang and Brahim Chaib-draa, editors, *Proceedings of the 16th Australian Conference on Artificial Intelligence (AI 2003)*, volume 2671 of *Lecture Notes in Computer Science*, pages 995–995. Springer Berlin / Heidelberg, 2003.

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In Barbara Hayes-Roth and Richard E. Korf, editors, *Proceedings of the 12th National Conference on Artificial Intelligence - AAAI'94*, volume 1, pages 337–343. AAAI Press / The MIT Press, 1994.

[SLM92] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In William R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992*, pages 440–446. AAAI Press / The MIT Press, 1992.

[SZ04] Haiou Shen and Hantao Zhang. Study of lower bound functions for max-2-sat. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004)*, pages 185–190. AAAI Press, 2004.

[SZ05] Haiou Shen and Hantao Zhang. Improving exact algorithms for max-2-sat. *Annals of Mathematics and Artificial Intelligence*, 44 :419–436, 2005.

[Tai91] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4–5) :443–455, 1991.

[TBSP02] John Thornton, Stuart Bain, Abdul Sattar, and Duc Nghia Pham. A two level local search for max-sat problems with hard and soft constraints. In *AI 2002 : Advances in Artificial Intelligence, 15th Australian Joint Conference on Artificial Intelligence, Canberra, Australia, December 2-6, 2002, Proceedings*, pages 603–614, 2002.

[TH03] Dave Tompkins and Holger Hoos. Scaling and probabilistic smoothing : Dynamic local search for unweighted max-sat. In Yang Xiang and Brahim Chaib-draa, editors, *Advances in Artificial Intelligence*, volume 2671 of *Lecture Notes in Computer Science*, pages 996–996. Springer Berlin / Heidelberg, 2003.

[TPBJ04] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for sat. 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 191–196, 2004.

[TS02] Orestis Telelis and Panagiotis Stamatopoulos. Heuristic backbone sampling for maximum satisfiablity. In *In Proceedings of the 2nd Hellenic Conference on Artificial Intelligence*, pages 129–139, 2002.

[Tse68] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II :115–125, 1968.

[US94] Tomás E. Uribe and Mark E. Stickel. Ordered binary decision diagrams and the davis-putnam procedure. In *Proceedings of the First International*

Conference on Constraints in Computational Logics, CCL '94, pages 34–49, London, UK, UK, 1994. Springer-Verlag.

[VG11] Allen Van Gelder. Generalized conflict-clause strengthening for satisfiability solvers. In Karem Sakallah and Laurent Simon, editors, *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, volume 6695 of *Lecture Notes in Computer Science*, pages 329–342. Springer Berlin / Heidelberg, 2011.

[vMvN05] Hans van Maaren and Linda van Norden. Sums of squares, satisfiability and maximum satisfiability. In Fahiem Bacchus and Toby Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 294–308. Springer Berlin / Heidelberg, 2005.

[WF96] Richard Wallace and Eugene C. Freuder. Comparative studies of constraint satisfaction and davis-putnam algorithms for maximum satisfiability problems. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability : Proceedings of the Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 587–615. American Mathematical Society, 1996.

[WW99] Zhe Wu and Benjamin W. Wah. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence - AAAI/IAAA'99*, pages 673–678. AAAI Press / The MIT Press, 1999.

[WW00] Zhe Wu and Benjamin W. Wah. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 310–315, 2000.

[XRS02] Hui Xu, Rob A. Rutenbar, and Karem Sakallah. sub-sat : a formulation for relaxed boolean satisfiability with applications in routing. In *Proceedings of the 2002 International Symposium on Physical Design (ISPD 2002)*, pages 182–187. ACM, 2002.

[XZ05] Zhao Xing and Weixiong Zhang. Maxsolver : An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2) :47–80, 2005.

[Yan92] Mihalis Yannakakis. On the approximation of maximum satisfiability. In Greg N. Frederickson, editor, *Proceedings of the 3rd Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA 1992)*, pages 1–9. ACM/SIAM, 1992.

[Yan94] Mihalis Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3) :475–502, 1994.

[YI98] Mutsunori Yagiura and Toshihide Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the max sat. In Wen-Lian Hsu and Ming-Yang Kao, editors, *Proceedings of the 4th Annual International Conference on Computing and Combinatorics (COCOON 1998)*, volume 1449 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 1998.

[YI01] M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the max sat : Experimental evaluation. *Journal of Heuristics*, 7 :423–442, 2001.

[ZLMA12] Zhu Zhu, Chu Min Li, Felip Manyà, and Josep Argelich. A new encoding from minsat into maxsat. In Michela Milano, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 455–463. Springer Berlin Heidelberg, 2012.

[ZRL03] Weixiong Zhang, Ananda Rangan, and Moshe Looks. Backbone guided local search for maximum satisfiability. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence - IJCAI'03*, pages 1179–1186. Morgan Kaufmann, 2003.

Résumé

Cette thèse porte sur la résolution du problème d'optimisation *Maximum Satisfiability* (Max-SAT). Nous y étudions en particulier les mécanismes liés à la détection et à la transformation des sous-ensembles inconsistants par la règle de la max-résolution. Dans le contexte des solveurs de type séparation et évaluation, nous présentons plusieurs contributions liées au calcul de la borne inférieure. Cela va du schéma d'application de la propagation unitaire utilisé pour détecter les sous-ensembles inconsistants à l'extension des critères d'apprentissage et à l'évaluation de l'impact des transformations par max-résolution sur l'efficacité des solveurs. Nos contributions ont permis l'élaboration d'un nouvel outil de résolution compétitif avec les meilleurs solveurs de l'état de l'art. Elles permettent également de mieux comprendre le fonctionnement des méthodes de type séparation et évaluation et apportent des éléments théoriques pouvant expliquer l'efficacité et les limites des solveurs existants. Cela ouvre de nouvelles perspectives d'amélioration, en particulier sur l'augmentation de l'apprentissage et la prise en compte de la structure interne des instances. Nous présentons également un exemple d'utilisation de la règle de la max-résolution dans un algorithme de recherche local.

Mots clés Max-SAT, Séparation et évaluation, Règles d'inférence, Max-résolution, Apprentissage, Recherche locale.

Abstract

This PhD thesis is about solving the Maximum Satisfiability (Max-SAT) problem. We study the mechanisms related to the detection and transformations of the inconsistent subsets by the max-resolution rule. In the context of the branch and bound (BnB) algorithms, we present several contributions related to the lower bound computation. They range from the study of the unit propagation scheme used to detect inconsistent subsets to the extension of the learning criteria and to the evaluation of the impact of the max-resolution transformations on the BnB solvers efficiency. Thanks to our contributions, we have implemented a new solver which is competitive with the state of art ones. We give insights allowing a better understanding of the behavior of BnB solvers as well as theoretical elements which contribute to explain the efficiency of these solvers and their limits. It opens new development perspectives on the learning mechanisms used by BnB solvers which may lead to a better consideration of the instances structural properties. We also present an example of integration of the max-resolution inference rule in a local search algorithm.

Keywords Max-SAT, Branch and Bound, Inference Rules, Max-résolution, Learning, Local Search.