UNIVERSITÉ D'ARTOIS

De la Satisfiabilité Propositionnelle aux Formules Booléennes Quantifiées

THÈSE

Soutenue le 5 décembre 2008

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Saïd Jabbour

Composition du jury

Rapporteurs :	Belaïd Benhamou (Maître de conférences, HdR)	Université de Provence
	Arnaud Lallouet (Professeur des Universités)	Université de Caen
Examinateurs :	Gilles Audemard (Maître de conférences)	IUT de Lens (Co-directeur)
	Chu Min Li (Professeur des Universités)	Université de Picardie Jules Vernes
	Éric Grégoire (Professeur des Universités)	Université d'Artois
	Youssef Hamadi (Chercheur)	Microsoft Research Cambridge
	Lakhdar Saïs (Professeur des Universités)	Université d'Artois (Directeur)

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

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

Remerciements

J’adresse mes premiers remerciements aux rapporteurs, messieurs Belaïd Benhamou et Arnaud Lallouet, qui, grâce à leurs remarques et conseils, m’ont permi d’améliorer ce rapport de thèse.

Je remercie également le professeur Chu Min Li pour m’avoir fait l’honneur d’examiner ce travail.

J’adresse un grand merci pour monsieur Éric Grégoire pour sa lecture attentive de ma thèse et pour toutes les remarques prodiguées ainsi que son aide précieuse pour résoudre des problèmes d’ordre administratifs que j’ai rencontré durant mon séjour.

Merci à monsieur Youssef Hamadi de m’avoir acceuilli à Microsoft Research Cambridge ainsi qu’à son équipe : Lucas Bordeaux et Horst Samulowitz avec qui j’ai partagé de bons moments et pour toutes les discussions menées sur pleins de sujets.

Je remercie infiniment mon directeur de thèse monsieur Lakhdar Saïs et mon co-directeur de thèse monsieur Gilles Audemard pour leur encadrement exemplaire et pour leurs conseils avisés, et leurs remarques souvents pertinentes.

Merci à tous les membres du CRIL pour leurs disponibilités et pour la bonne ambiance reignante au sein du laboratoire ainsi qu’à tous les doctorants avec qui j’ai passé de bons moments à discuter sur pleins de sujets de recherche et autres.

Je remercie également le conseil régional Nord-pas-de-calais pour son soutien financier, très important pour mener à bien ce travail.

Un grand Merci à ma famille pour son soutien et sa confiance, plus particulièrement à ma soeur Ytto et à mon frère Mustapha.

Merci à Pauline pour sa patience et pour son soutient et d’avoir supporter mes longues absences.

Un grand merci à mes amis de Nancy Youssef Moumadah et Samir El-Goufiri pour leurs amitiés ainsi qu’à Bassim Sefiani Fahd de Lille.

Je dédie cette thèse
à ma famille

Table des matières

Table des figures **ix**

Liste des tableaux **x**

Liste des algorithmes **xi**

Introduction générale **xii**

Partie I SAT

Chapitre 1 Logique Propositionnelle & SAT **1**

1.1 Logique propositionnelle 1

1.1.1 Syntaxe 1

1.1.2 Sémantique 2

1.1.3 Formes normales 3

1.2 Problème SAT 4

1.2.1 Définition du problème 4

1.2.2 Classes polynomiales 4

1.2.3 Algorithmes de résolution de SAT 5

1.3 Problèmes autour de SAT 16

Chapitre 2 Un cadre étendu pour l'analyse de conflits **18**

2.1 Introduction 18
2.2 Quelques notations préliminaires 19
2.3 Analyse des conflits : approche classique (CDCL) 20
2.3.1 Graphe d'implications 20
2.3.2 Génération des clauses assertives 21
2.4 Preuve d'optimalité du schéma "First UIP" 23
2.5 eCDCL : un cadre étendu pour l'analyse de conflits 24
2.5.1 Graphe d'implications étendu 24
2.5.2 Génération de clauses assertives étendues 26
2.6 Mise en œuvre en pratique 28
2.6.1 Améliorer la qualité du saut arrière 28
2.6.2 Réduire la taille des clauses assertives 28
2.7 Expérimentations 29
2.8 Conclusion 30

Chapitre 3 Apprentissage à partir des succès **34**

3.1 Introduction 34
3.2 Apprentissage à partir des succès 35
3.2.1 Motivation 35
3.2.2 Présentation Formelle 36
3.3 Intégration de l'approche SDCL dans un solveur SAT moderne 37
3.4 Expérimentations 39
3.5 Conclusion 43

Chapitre 4 Résolution parallèle de SAT **44**

4.1 Introduction 44
4.2 Stratégies dans les solveurs SAT modernes 45
4.3 Architecture multicore 46
4.4 ManySAT : un solveur SAT parallèle 46
4.4.1 Politique de redémarrage 46
4.4.2 Heuristiques 47
4.4.3 Polarité 47
4.4.4 Apprentissage 47
4.4.5 Partage de clauses 48

4.5 Evaluation 48
4.6 Travaux précédents 48
4.7 Conclusion 50

Chapitre 5 Graphe SAT : une nouvelle forme de représentation **52**
5.1 Introduction 53
5.2 Graphe-SAT : une nouvelle représentation de formules CNF 53
5.3 Pontages algorithmiques : CNF / Graphe-SAT 55
5.3.1 Résolution & Fermeture Transitive 55
5.4 Graphe-SAT : Applications 57
5.4.1 Ensembles 2-SAT strong backdoors 57
5.4.2 Génération d'instances SAT difficiles 58
5.4.3 Graphe & pré-traitement 62
5.5 Expérimentations 64
5.5.1 Calcul d'ensembles 2-SAT Strong Backdoors 64
5.5.2 Évaluation du Générateur Diamond 67
5.5.3 Évaluation du pré-traitement 69
5.6 Conclusion 73

Partie II QBF

Chapitre 6 Formules Booléennes Quantifiées **76**
6.1 Syntaxique 76
6.2 Sémantique 79
6.3 Règles de simplification 80
6.4 Algorithmes de résolution pour QBF 83
6.4.1 DPLL pour QBF 83
6.4.2 Apprentissage 84
6.5 Transformations : de QBF vers SAT 87
6.5.1 Élimination de variables dans les QBF 88
6.5.2 Résolution et extension 88

6.5.3 Élimination symbolique des quantificateurs 89
6.5.4 QBF et fonctions de skolem 90
6.5.5 QBF et codage en BDD 92

Chapitre 7 Symétries et QBFs **94**
7.1 Introduction 95
7.1.1 Symétries et SAT 96
7.1.2 Symétries et QBF 98
7.2 Détection de symétries : de SAT à QBF 99
7.3 Suppression des symétries dans les QBF 100
7.4 Approche basée sur les modèles (M-QSB) 102
7.4.1 M-QSB : Transformation générale 102
7.4.2 M-QSB : Transformation clausale 105
7.5 Approche basée sur les nogoods (N-QSB) 108
7.5.1 N-QSB : Élimination d'une seule symétrie 109
7.5.2 N-QSB : Élimination d'un ensemble de symétries 112
7.5.3 N-QSB : Transformation du préfixe 114
7.5.4 Out-littéraux positifs et négatifs 114
7.5.5 Complexité 115
7.6 Expérimentations 115
7.6.1 Résumé des résultats 116
7.6.2 Comparaison des résultats 117
7.7 Conclusion 117
7.8 Travaux futurs 122

Conclusion générale **123**

Bibliographie **126**

Table des figures

1.1 Retour arrière chronologique 7
1.2 Quine vs DPLL 9
1.3 watched literals 16

2.1 Graphe d'implication $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ 21
2.2 Graphe d'implication $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ 23
2.3 Graphe d'implication étendu : $\mathcal{G}s^{\rho}_{\mathcal{F}'} = (\mathcal{N}', \mathcal{E}')$ 25
2.4 Résultats sur les instances de la compétition SAT'07 (industriel) et de la Sat-Race'06 . . 31
2.5 Temps et temps cumulé pour résoudre un nombre donné d'instances 32

3.1 Graphe d'implication $\mathcal{G}^{\rho}_{\mathcal{F}}$ de l'exemple 6 35
3.2 CDCL Vs SDCL 40
3.3 Instances satisfiables 41
3.4 Instances insatisfiables 42

4.1 Stratégies de redémarrage 47
4.2 Différentes limites pour la taille des clauses échangées 48
4.3 ManySAT avec et sans échange 50

5.1 Graphe SAT $G_{\mathcal{F}}$ associé la formule $\mathcal{F}$ 55
5.2 Une hyper-formule étendue B^m_3 et l'ensemble de clauses associé 60
5.3 Formule diamond $\mathcal{F}^m_l(a)$: une hiérarchie d'hyper-formules étendues 61
5.4 Graphe SAT partiel $G^p_{\mathcal{F}}$ associé à la formule $\mathcal{F}$ (exemple 10) 63
5.5 Hyper-Res générale /Hyper-Res étendue/ Hyper-Res / Hyper-Res Bin : une comparaison 64
5.6 Pourcentage d'instances résolues par rapport au niveau l 67
5.7 Temps de résolution ($m = 4$) 68
5.8 Temps de résolution($m = 5$) 68
5.9 Comparaison des temps de calcul avec et sans pré-traitement sur les formules diamond . 70
5.10 Nombre d'instances résolues par rapport au temps CPU 71

6.1 Politique totale de la QCNF $\mathcal{F}$ 80

7.1 Réduction du graphe 100
7.2 Arbre de recherche de $\mathcal{F}^m$ 105
7.3 Arbre de recherche (exemple 30) 107
7.4 L'approche $N - QSB$: un exemple illustratif 109
7.5 Graphe de précédence de σ_j 111
7.6 Arbre de recherche $\mathcal{F}^n$ de l'exemple 20 112
7.7 Graphe de précédence de $\sigma_1[X_1]$ et $\sigma_2[X_1]$ (exemple 32) 114

Liste des tableaux

2.1 Quelques résultats comparatifs 33

3.1 Un aperçu des résultats 43

4.1 ManySAT (SAT-Race 2008) 49

5.1 2-SAT vs Horn strong backdoors 66
5.2 Petites instances SAT difficiles ($m = 4, 5$) 69
5.3 Petites instances UNSAT difficiles ($m = 4, 5$) 69
5.4 Comparaison entre MINISAT, MINISAT + pré-traitement et MINISAT + SatELite 72

7.1 Familles d'instances 116
7.2 Résultats obtenus avec NCQUBE 118
7.3 Résultats obtenus avec SKIZZO 119
7.4 Résultats obtenus avec SQBF 120
7.5 Résultats obtenus avec SEMPROP 121

Liste des algorithmes

1 Algorithme DP 6
2 Algorithme DPLL 10
3 Algorithme CDCL 13
4 Procédure SDCL 38
5 Learn_from_success 39
6 Approximation d'un ensemble 2-SAT strong backdoors 59
7 HypResGen : une étape du pré-traitement 65
8 HyReGenPre : un pré-traitement par hyper-résolution générale 65
9 Algorithme $\mathcal{Q} - \mathcal{DPLL}$ pour QBF 84
10 Algorithme $\mathcal{Q} - \mathcal{CDCSL}$ avec apprentissage 86
11 Multi-Résolution pour QBF en utilisant des $\mathcal{ZBDD}$ 90

Introduction générale

Depuis plusieurs années, le problème de la satisfiabilité propositionnelle (décider si une formule booléenne mise sous forme normale conjonctive admet ou non une valuation qui la rend vraie) a fait l'objet de nombreux travaux ayant abouti à des progrès remarquables sur le double plan théorique et pratique. Sur le plan théorique les découvertes du double phénomène du seuil et de longue traîne ("heavy tail") respectivement sur les instances aléatoires et industrielles constituent sans aucun doute les résultats les plus importants permettant d'affiner notre compréhension de la difficulté relative de SAT et d'expliquer les progrès récents dans la résolution pratique de ce problème. Le premier met en évidence là où les instances sont les plus difficiles. En particulier, la résolution d'instances aléatoires insatisfiables générées dans la phase de transition constitue à l'heure actuelle un vrai défi pour les algorithmes de SAT (cf. challenge 1 [SELMAN *et al.* 1997]). Le second met en évidence l'existence de preuves courtes pour de nombreuses instances industrielles. En d'autres termes, différentes exécutions (ou ordres des variables) du même algorithme peuvent conduire à des variations importantes en temps d'exécution. Aujourd'hui, résoudre une instance aléatoire insatisfiable au seuil de 700 variable et 2975 clauses nécessiterait un temps d'exécution prohibitif qui se chiffre à plusieurs journées de temps CPU. Alors que la résolution efficace d'instances industrielles de plusieurs centaines de milliers de variables et de millions de clauses par les solveurs SAT modernes est maintenant une réalité. En effet, au-delà des domaines d'applications traditionnels de SAT comme la vérification formelle de circuits, de nouvelles applications issues de domaines variés comme la cryptographie, la configuration de produits, la planification, et même la bio-informatique sont maintenant résolues par des approches SAT.

Ce bref aperçu démontre clairement que le caractère NP-Complet du problème SAT reste un verrou théorique majeur, mais ne constitue nullement un obstacle à la mise en œuvre de techniques algorithmiques efficaces en pratique. Cette évolution historique dans la résolution de SAT a contribué au regain d'intérêt dans la résolution de problèmes autour de SAT encore plus difficiles. Le problème de la satisfiabilité maximale (Max-SAT), le comptage du nombre de solutions (#SAT) et la résolution de formules booléennes quantifiées (QBF ($PSPACE - Complet$)) en font partie. Ce dernier problème qui est défini comme une généralisation de SAT où les variables sont quantifiées existentiellement ($\exists$) ou universellement ($\forall$) est plus difficile que SAT et occupe lui aussi une place centrale dans cette classe de complexité. Le problème QBF se définit donc comme une extension naturelle de SAT. Au-delà des aspects théoriques suscités par la recherche d'algorithmes pour le résoudre, ces applications dans plusieurs domaines (planification, raisonnement non monotone, vérification formelle, diagnostic etc.) ont motivé son étude. Ces motivations ne s'arrêtent pas là, car les progrès fulgurant réalisés dans la résolution du problème SAT constituent une base non négligeable pour la résolution du problème QBF. L'adaptation des algorithmes de SAT en QBF a permis une évolution et un regain d'intérêt pour ce problème. Il reste que la taille des instances QBF résolues est comparable à celle des instances SAT résolues il y a une dizaine d'années.

Finalement pour dresser un bilan, les solveurs SAT et QBFs ont su se moderniser. Malgré les résultats négatifs de complexité, la résolution de ces deux problèmes a connu des avancés importantes. Il demeure que de nouveaux pas doivent être franchis pour permettre la résolution d'instances encore plus difficiles et plus grandes.

C’est dans ce contexte que s’inscrit cette thèse, notre objectif est de proposer de nouvelles approches originales pour élargir la classe des problèmes que l’on peut résoudre en pratique pour ces deux problèmes fondamentaux SAT et QBF.

La thèse est organisée en deux parties.

Dans la première partie, après un rappel de la logique propositionnelle et SAT, nous donnerons une description des principales approches de résolution du problèmes SAT en relation direct avec nos contributions dans le cadre de cette thèse.

Le deuxième chapitre présente plusieurs contributions au schéma CDCL ("*Conflict Driven Clauses Learning*"), une des composantes clés des solveurs SAT modernes (voir section 2.3). Tout d’abord, nous montrons qu’à partir du graphe d’implications, les clauses assertives obtenues en utilisant le principe du premier point d’implication unique "*First Unique Implication Point*" (First UIP) sont optimales en terme de sauts ou de retours-arrière. Puis nous proposons une extension [AUDEMARD *et al.* 2008a] du graphe d’implications contenant des arcs additionnels appelés arcs inverses. Ces arcs sont obtenus en tenant compte des clauses satisfaites qui sont habituellement ignorées par l’analyse du conflit. Cette extension capture plus fidèlement l’ensemble du processus de propagation et ouvre de nouvelles perspectives pour les approches fondées sur CDCL. Entre autres avantages, notre extension du graphe d’implication conduit à un nouveau schéma d’analyse des conflits qui exploite les arcs ajoutés et permet des retours-arrière plus haut dans l’arbre de recherche. Les résultats expérimentaux montrent que l’intégration de notre schéma d’analyse des conflits généralisé au sein de solveurs les plus efficaces améliore sensiblement leurs performances.

Dans le troisième chapitre, un nouveau schéma d’apprentissage pour SAT est proposé. L’originalité de cette approche réside dans sa capacité à opérer de l’apprentissage à chaque nœud de l’arbre de recherche et pas uniquement lors d’un conflit. Ceci contraste avec toutes les approches d’apprentissage traditionnelles qui généralement se réfèrent à l’analyse de conflit. Pour rendre cet apprentissage possible, des clauses pertinentes, extraites à partir de la partie satisfaite de la formule, sont utilisées conjointement avec le graphe d’implications classique pour dériver de nouvelles explications plus puissantes pour un littéral donnée. Basé sur cette extension, un premier schéma appelé apprentissage à partir des succès “Success Driven Clause Learning” (SDCL) est proposé. Des résultats expérimentaux montrent que l’intégration de SDCL dans un solveur SAT moderne comme Minisat permet d’obtenir des résultats intéressants particulièrement sur les instances satisfiables.

Dans le quatrième chapitre, un nouveau solveur parallèle ManySAT pour une architecture multicore est présenté. La conception de ManySAT tire parti des principales faiblesses des solveurs SAT modernes, à savoir leurs sensibilités aux réglages des paramètres et leurs manques de robustesse. ManySAT utilise un portfolio complémentaire d’algorithmes séquentiels conçus grâce à une minutieuse variation dans les principales stratégies de résolution. Ces stratégies orthogonales sont combinées au partage de clauses dans le but d’améliorer les performances globales de ManySAT. Ceci contraste avec la majorité des solveurs parallèles existants, qui généralement utilisent le paradigme "diviser pour régner". Les résultats des expérimentations sur plusieurs classes d’instances SAT industrielles, et le premier rang (“gold medal”) obtenu par ManySAT lors de la compétition SAT Race 2008 organisée en Chine, montrent clairement le potentiel qu’il y a derrière la philosophie de conception de notre solveur parallèle.

Nous proposons dans le cinquième chapitre une nouvelle représentation sous forme de graphe d’une formule CNF [AUDEMARD *et al.* 2008b]. Elle étend le graphe d’implications généralement utilisé dans le cas des clauses binaires (2-SAT) au cas général. Chaque clause est représentée comme un ensemble d’implications conditionnelles. Dans notre cas, par implication conditionnelle on exprime l’assertion sui-

vante : un littéral a implique b sous la condition C où C est une conjonction de littéraux. Dans le cas 2-SAT, C est un ensemble vide. Cette notion d'implication conditionnelle, nous permet pour des clauses de taille quelconque de rester dans le cadre d'une représentation en termes de graphes valués, au lieu d'une représentation en hypergraphes, généralement plus difficile à manipuler. Une clause est donc exprimée par une implication conditionnelle qui est elle même codée par un arc étiqueté par un ensemble de littéraux, appelé *contexte, ou condition*. Cette nouvelle représentation permet d'étendre quelques propriétés algorithmiques intéressantes utilisées dans le cadre du fragment traitable 2-SAT (voir la section 5.4.1). Parmi elles, la résolution classique est reformulée en utilisant la fermeture transitive d'un graphe. Les (plus courts) chemins entre les nœuds $\neg a$ et a donnent une façon originale de calculer les conditions (minimales) sous lesquelles un littéral a est impliqué. Cette nouvelle représentation offre de nombreux avantages et diverses utilisations intéressantes. Dans le cadre de ce chapitre, trois utilisations concrètes de cette représentation de formules CNF quelconques en graphe sont présentées et validées expérimentalement. La première concerne l'extraction des ensembles 2-SAT "strong backdoors". Dans la seconde, nous exploitons cette représentation pour la génération d'instances SAT difficiles. Alors que dans la troisième, nous dérivons une nouvelle technique de pré-traitement des formules CNF.

Dans la deuxième partie, le chapitre six présente une extension de la logique propositionnelle vers la logique QBF. Ensuite, nous ferons un tour d'horizon des algorithmes de résolution du problème QBF.

Dans le septième chapitre, nous proposons de supprimer les symétries dans les formules QBF. Si dans le cas SAT, le fameux SBP (symmetry breaking predicates) est le plus utilisé pour supprimer les symétries. Dans le cas QBF, nous proposons deux approches pour supprimer les symétries.

La première approche [AUDEMARD *et al.* 2007b] constitue une extension du SBP de SAT vers QBF. Nous montrons dans ce cadre le fondement de cette extension qui consiste, en plus du SBP classique, à ajouter de nouvelles contraintes appelées QSBP (quantified symmetry breaking predicates) et un nouvel ordre des variables universelles qui interviennent dans les symétries est calculé via un graphe appelé graphe de précédence.

La deuxième approche [AUDEMARD *et al.* 2007a] proposée dans ce chapitre peut être vue comme une approche duale de la première. En effet si dans le cadre SAT, les interprétations symétriques ne peuvent être considérées que comme des contre-modèles de la formule, dans le cas QBF et grâce à la présence des variables universelles, nous verrons que sous certaines conditions, nous pouvons considérer certaines interprétations symétriques comme des modèles de la formule. Dans ce cadre la formule est transformée en une formule CNF/DNF. La forme CNF de la formule est constituée de la matrice de la QBF originale augmentée par le SBP classique projeté sur les parties existentielles des symétries. Et la partie DNF est un ensemble de modèles prédéfini. Enfin une transformation de la CNF/DNF vers CNF est proposée. Des résultats expérimentaux sur un large panel d'instances symétriques montrent l'intérêt de supprimer les symétries dans le cas QBF.

Enfin, une conclusion générale et des nombreuses perspectives terminent ce mémoire.

Ce travail a donné lieu à une distinction :
– ManySAT, meilleur solveur parallèle ("Gold Medal"), SAT Race 2008, travail conjoint avec Youssef Hamadi et Lakhdar Saïs

et aux publications suivantes :
– Gilles Audemard, Saïd Jabbour, and Lakhdar Saïs. Graph-Based SAT Representation : A New Perspective. *Journal of Algorithms in Logic, Informatics and Cognition*, (63) 17-33, 2008.

– Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbour, and Lakhdar Saïs. A Generalized Framework for Conflict Analysis. In *(SAT'2008)*, H. Kleine Büning and X. Zhao (Eds.), LNCS 4996, pages 21-27, Guangzhou, Chine, 2008 (papier court).

– Youssef Hamadi, Saïd Jabbour, and Lakhdar Saïs. ManySAT : Solveur description. In *SAT race 2008*, `www-sr.informatik.uni-tuebingen.de/sat-race-2008/descriptions/solver_24.pdf`, 2008 (également en rapport interne MSR-TR-2008-83, Microsoft Research Cambridge, mai 2008).

– Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbbour, Lakhdar Saïs, Un cadre général pour l'analyse des conflits, dans *Actes des quatrièmes Journées Francophones de Programmation par Contraintes (JFPC'08)*, pages 267–276, juin 2008.

– Gilles Audemard, Saïd Jabbour, and Lakhdar Saïs. Symmetry breaking in quantified boolean formulae. In *(IJCAI'07)*, pages 2262-2267, Hyderabad, Inde, 2007.

– Gilles Audemard, Saïd Jabbour, Lakhdar Saïs. Efficient Symmetry breaking Predicates for quantified boolean formulae. In *Proceedings of the International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'07) - Affiliated to CP* , pages 14–21, Povidence, USA, 2007.

– Gilles Audemard, Saïd Jabbour, Lakhdar Saïs. Using SAT-Graph Representation to Derive Hard SAT instances. In *Proceedings of the 14th Intl. Workshop on Expeimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'07)*, Rome, 5-6 juillet 2007.

– Saïd Jabbour, SAT-Graph-Based Representation : a new perspective. Dans *Proceedings des Journées d'Intelligence Artificielles Fondamentale*, 2-3 juillet 2007, Grenoble 2007.

– Gilles Audemard, Saïd Jabbour, Lakhdar Saïs. Exploitation des symétries dans les formules booléennes quantifiées. Dans *Journées Francophones de la Programmation par Contraintes (JFPC'06)*, pages 15-24. Nîmes 2006.

Première partie

SAT

1

Logique Propositionnelle & SAT

Sommaire

1.1 Logique propositionnelle 1
1.1.1 Syntaxe 1
1.1.2 Sémantique 2
1.1.3 Formes normales 3
1.2 Problème SAT 4
1.2.1 Définition du problème 4
1.2.2 Classes polynomiales 4
1.2.3 Algorithmes de résolution de SAT 5
1.3 Problèmes autour de SAT 16

1.1 Logique propositionnelle

1.1.1 Syntaxe

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

Définition 2 (langage propositionnel) *Soit* $\mathcal{P}$ *un ensemble fini de symboles propositionnels appelés également variables ou atomes. Pour chaque sous-ensemble* $\mathcal{V}$ *de* $\mathcal{P}$, $\mathcal{P}_\mathcal{V}$, *désigne le langage propositionnel construit à partir des symboles de* $\mathcal{V}$, *des parenthèses "(" et ")", constantes booléennes faux (* $\top$ *) et vrai (* $\bot$ *), et des connecteurs logiques* $\neg$ *(non),* $\wedge$ *(et),* $\vee$ *(ou),* $\rightarrow$ *(implique),* $\leftrightarrow$ *(équivalence). Une suite finie d'éléments du langage est une expression.*

Définition 3 (formule propositionnelle) *L'ensemble des formules propositionnelles est le plus petit ensemble d'expressions tel que*

1. *un atome,* $\top$ *et* $\bot$ *sont des formules ;*
2. *si* $\mathcal{F}$ *est une formule alors* $\neg\mathcal{F}$ *est une formule ;*
3. *si* $\mathcal{F}$ *et* $\mathcal{F}'$ *sont des formules alors*
   (a) $(\mathcal{F} \wedge \mathcal{F}')$ *est une formule ;*
   (b) $(\mathcal{F} \vee \mathcal{F}')$ *est une formule ;*
   (c) $(\mathcal{F} \rightarrow \mathcal{F}')$ *est une formule ;*

(d) ($\mathcal{F} \leftrightarrow \mathcal{F}'$) est une formule ;

Définition 4 (littéral) *Un littéral est une variable propositionnelle x (littéral positif) ou sa négation $\neg x$ (littéral négatif). Les deux littéraux x et $\neg x$ sont dits complémentaires. On note également $\neg l$ (ou $\bar{l}$) le littéral complémentaire de l.*

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

1.1.2 Sémantique

Définition 5 (interprétation) *Une interprétation ρ en calcul propositionnel est une application associant à toute formule propositionnelle $\mathcal{F}$ une valeur $\rho(\mathcal{F})$ dans $\{faux, vrai\}$. L'interprétation $\rho(\mathcal{F})$ d'une formule $\mathcal{F}$ est définie par la valeur de vérité donnée à chacun des atomes de $\mathcal{F}$. $\rho(\mathcal{F})$ se calcule par l'intermédiaire des règles suivantes :*

1. $\rho(\top) = vrai$;
2. $\rho(\bot) = faux$;
3. $\rho(\neg\mathcal{F}) = vrai$ ssi $\rho(\mathcal{F}) = faux$;
4. $\rho(\mathcal{F} \wedge \mathcal{G}) = vrai$ ssi $\rho(\mathcal{F}) = \rho(\mathcal{G}) = vrai$;
5. $\rho(\mathcal{F} \vee \mathcal{G}) = faux$ ssi $\rho(\mathcal{F}) = \rho(\mathcal{G}) = faux$;
6. $\rho(\mathcal{F} \rightarrow \mathcal{G}) = faux$ ssi $\rho(\mathcal{F}) = vrai$ et $\rho(\mathcal{G}) = faux$;
7. $\rho(\mathcal{F} \leftrightarrow \mathcal{G}) = vrai$ ssi $\rho(\mathcal{F}) = \rho(\mathcal{G})$.

L'interprétation ρ d'une formule $\mathcal{F}$ est dite totale ssi $\forall x \in \mathcal{V}(\mathcal{F})$, on a soit $\rho(x) = vrai$, soit $\rho(x) = faux$; elle est dite partielle sinon. On représente souvent une interprétation ρ comme un ensemble de littéraux, i.e, $x \in \rho$ si $\rho(x) = vrai$, $\neg x \in \rho$ si $\rho(x) = faux$.

On note $S_\rho(\mathcal{F})$ l'ensemble des interprétations d'une formule $\mathcal{F}$. Si $\mathcal{F}$ possède exactement n variables propositionnelles alors $|S_\rho(\mathcal{F})|=2^n$.

Définition 6 (modèle) *Une interprétation ρ satisfait une formule $\mathcal{F}$ si $\rho(\mathcal{F}) = vrai$. On dit alors que ρ est un modèle de $\mathcal{F}$, et qu'on note $\rho \models \mathcal{F}$. Inversement, si $\rho(\mathcal{F}) = faux$, on dit que ρ falsifie $\mathcal{F}$ et que ρ est un contre-modèle de $\mathcal{F}$ (ou nogood), et qu'on note communément $\rho \models \neg\mathcal{F}$.*

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

Définition 7 (impliquant premier) *Soit $\mathcal{F}$ une formule CNF et ρ un modèle de $\mathcal{F}$. ρ est appelé impliquant premier de $\mathcal{F}$ si pour tout $x \in \rho$, $\rho - \{x\}$ n'est pas un modèle de $\mathcal{F}$*

Définition 8 (formule satisfiable) *Une formule propositionnelle $\mathcal{F}$ est satisfiable si elle admet au moins un modèle ρ, i.e, $M_\rho(\mathcal{F}) \neq \emptyset$.*

Définition 9 (formule insatisfiable) *Une formule propositionnelle $\mathcal{F}$ est dite insatisfiable s'il n'existe pas d'interprétation ρ satisfaisant $\mathcal{F}$ i.e, $M_\rho(\mathcal{F}) = \emptyset$*

Définition 10 (tautologie) *Une formule propositionnelle $\mathcal{F}$ est une tautologie si toute interprétation de $\mathcal{F}$ est un modèle de $\mathcal{F}$ i.e, $M_\rho(\mathcal{F})= S_\rho(\mathcal{F})$.*

Définition 11 *Soient $\mathcal{F}$, $\mathcal{G}$ deux formules propositionnelles, on dit que $\mathcal{G}$ est conséquence logique de $\mathcal{F}$, notée $\mathcal{F} \models \mathcal{G}$, ssi $M_\rho(\mathcal{F}) \subseteq M_\rho(\mathcal{G})$. $\mathcal{F}$ et $\mathcal{G}$ sont équivalentes ssi $\mathcal{F} \models \mathcal{G}$ et $\mathcal{G} \models \mathcal{F}$*

Remarque 1 *De la définition d'une interprétation on déduit les équivalences suivantes : $(\mathcal{F} \rightarrow \mathcal{G}) \equiv (\neg\mathcal{F} \vee \mathcal{G})$ et $(\mathcal{F} \leftrightarrow \mathcal{G}) \equiv (\mathcal{F} \rightarrow \mathcal{G}) \wedge (\mathcal{G} \rightarrow \mathcal{F})$;*

1.1.3 Formes normales

Définition 12 (clause) *Une clause est une disjonction finie de littéraux. Une clause ne contenant pas de littéraux complémentaires est dite fondamentale, sinon elle est dite tautologique.*

Définition 13 (forme CNF) *Une formule $\mathcal{F}$ est sous **forme normale conjonctive (CNF)** si et seulement si $\mathcal{F}$ est une conjonction de clauses.*

$$\mathcal{F} = \bigwedge_{c \in \mathcal{F}} (\bigvee^{l_i \in c} l_i)$$

Il est naturel de considérer une formule CNF comme un ensemble de clauses et chaque clause comme un ensemble de littéraux.

Définition 14 (différentes variantes de clauses)

- ***(clause unitaire)*** *Une clause c est dite unitaire (mono-littéral) ssi elle contient seulement un seul littéral.*
- ***(clause binaire)*** *Une clause c est dite binaire ssi elle contient exactement deux littéraux.*
- ***(clause de Krom)*** *Une clause de Krom est une clause qui contient au plus deux littéraux.*
- ***(clause positive, négative, mixte)*** *On appelle clause **positive**(resp. une clause **négative**) une clause qui ne contient pas de littéraux négatifs (resp. positifs). Une clause constituée de littéraux positifs et négatifs est appelée clause **mixte**.*
- ***(clause vide)*** *Une clause c vide, notée $\perp$, est une clause ne contenant aucun littéral. Elle est par définition insatisfiable.*
- ***(clause tautologique)*** *Une clause c tautologique est une clause contenant un littéral et son complémentaire. Elle est par définition vraie.*
- ***(clause de Horn, clause reverse-Horn)*** *Une clause de Horn (resp. clause reverse Horn) est une clause qui contient au plus un littéral positif (resp. négatif).*

ρ est un modèle d'une formule CNF $\mathcal{F}$ si et seulement si chaque clause de $\mathcal{F}$ est satisfaite par au moins un littéral de ρ : $\forall c \in \mathcal{F}$, $\exists x \in (\mathcal{F})$ tel que $x \in c$ et $x \in \rho$.

Définition 15 (forme DNF) *Une formule $\mathcal{F}$ est sous **forme normale disjonctive(DNF)** si et seulement si $\mathcal{F}$ est une disjonction de monômes. Un monôme est une conjonction de littéraux.*

$$\mathcal{F} = \bigvee^{\rho \in \mathcal{F}} (\bigwedge_{l_i \in \rho} l_i)$$

1.2 Problème SAT

1.2.1 Définition du problème

Définition 16 (problème SAT) *Soit $\mathcal{F}$ une formule CNF. Le problème SAT est un problème de décision qui consiste à déterminer si $\mathcal{F}$ admet ou non un modèle.*

Exemple 1 *Donnée : Une formule booléenne mise sous forme CNF :*

$$\mathcal{F} = \begin{cases} (x_1 \vee x_2 \vee x_3) & \wedge \\ (\neg x_1 \vee x_2) & \wedge \\ (\neg x_2 \vee x_3) & \wedge \\ (\neg x_3 \vee x_1) & \end{cases}$$

Question : Est-ce que la formule $\mathcal{F}$ admet au moins un modèle ?
Réponse : Pour cet exemple, la réponse est oui : l'affectation $x_1 = vrai, x_2 = vrai, x_3 = vrai$ satisfait la formule $\mathcal{F}$. Comme $\mathcal{F}$ admet une seul modèle, on a $\mathcal{F} \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ est insatisfiable.

Le problème SAT est le problème NP-complet de référence [COOK 1971]. Par sa simplicité, la forme CNF est généralement la forme normale standard des solveurs SAT. En effet, de nombreux problèmes s'expriment naturellement comme une conjonction de contraintes simples : les clauses. La technique de transformation de Tseitin [TSEITIN 1968], peut être utilisée pour transformer linéairement toute formule propositionnelle en une formule CNF équivalente pour la satisfiabilité, grâce à l'introduction de variables supplémentaires pour représenter les sous-formules. De la même manière, cet ajout de nouvelles variables peut aussi servir à transformer une formule CNF en une formule $3-SAT$ dont les clauses sont toutes de longueur 3 (une restriction de SAT qui reste NP-complète).

Vu le nombre important de travaux de recherche réalisé depuis de nombreuses années sur SAT, il est très difficile, voir impossible, d'effectuer un inventaire complet. Ce qui dépasserait le cadre de cette thèse. Je me limiterai à une description des travaux en relation directe avec mes contributions de recherche. Le lecteur intéressé par un panorama plus complet peut se reporter au récent livre traitant du problème SAT [SAIS 2008].

1.2.2 Classes polynomiales

Une classe polynomiale est un ensemble de formules qu'il est possible de reconnaître et de résoudre en temps polynômial. Une classe polynomiale admet d'autant plus d'intérêt qu'elle recouvre une large classe de problèmes réels. Outre l'intérêt théorique de ce type d'étude, un autre bénéfice concerne l'exploitation de ces classes pour mieux traiter le cas général [GALLO & URBANI 1989, BACCHUS 2002].
Parmi les « nombreuses » classes polynomiales du problème SAT, 2-SAT [COOK 1971] (instances formées de clauses binaires) et Horn-SAT (instances formées de clauses ayant au plus un littéral positif) restent sans aucun doute parmi les plus intéressantes. En effet, les clauses binaires se retrouvent très fréquemment dans les problèmes réels, un traitement particulier sur les clauses binaires améliore considérablement l'efficacité des algorithmes de résolution. On sait aussi que les clauses de Horn sont à la base des langages de programmation logique, etc. On retrouve ensuite les formules Horn-renommables (des formules qui se réduisent à Horn par un renommage adéquat des littéraux) et les q-Horn (une généralisation de Horn-SAT et 2-SAT). Il existe bien d'autres classes polynomiales comme par exemple celles exhibées par l'introduction de différentes formes de hiérarchies d'instances, par des restrictions sur le nombre d'occurrences etc.

1.2.3 Algorithmes de résolution de SAT

De nombreux algorithmes ont été proposés pour résoudre SAT. Il serait présomptueux de prétendre réaliser un inventaire complet de ces méthodes. Les principes à la base de ces méthodes sont très variés : résolution [ROBINSON 1965], réécriture [BOOLE 1854, SHANNON 1938], énumération [DAVIS *et al.* 1962], comptage du nombre de solutions [IWAMA 1989], recherche locale [SELMAN *et al.* 1992], programmation linéaire en nombres entiers [C.E. BLAIR & LOWE 1986, BENNACEUR 1995], diagrammes binaires de décision [AKERS 1978], algorithmes génétiques et évolutionnistes [LARDEUX *et al.* 2002], etc.

Preuve par résolution

L'une des règles les plus fondamentales de la logique propositionnelle est sans aucun doute la règle de résolution (Robinson [ROBINSON 1965]).

Définition 17 (résolvante) *Soient c_i et c_j deux clauses contenant respectivement x et $\neg x$, on définit la résolvante entre c_i et c_j sur x, notée $\eta[x, c_i, c_j]$, comme la clause $r = c_i \cup c_j - \{x, \neg x\}$.*

Propriété 1 *Soient $\mathcal{F}$ une formule, r la résolvante de c_i et c_j deux clauses de $\mathcal{F}$ contenant respectivement x et $\neg x$. Les deux propriétés suivantes sont vraies :*

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

La *règle de résolution* est le processus de dérivation d'une résolvante à partir de deux clauses contenant deux littéraux complémentaires.

Définition 18 (dérivation par résolution et réfutation) *Une dérivation par résolution d'une clause c à partir de $\mathcal{F}$ est une séquence $\pi = [c_1, c_2, \ldots, c_k = c]$ et $\forall 1 < i \leq k$, soit (1) $c_i \in \mathcal{F}$, soit (2) $c_i = \eta[x, c_k, c_j]$ avec $1 \leq k, j < i$.*

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

La résolution est un système de preuve complet pour la réfutation.

Définition 19 ***(subsumption)*** *Une clause c_1 subsume c_2 ssi $c_1 \subseteq c_2$. On a aussi $c_1 \vDash c_2$.*

Une autre règle souvent omise, appelée *fusion* consiste à remplacer les multiples occurrences d'un littéral par une seule occurrence de ce littéral.

L'application de la subsumption et de la fusion à chaque étape de la résolution rend le système complet. Si la formule est insatisfiable, on obtient une clause vide ; sinon, un point fixe (toute nouvelle résolvante est subsumée par une clause existante) est atteint, et la formule est donc satisfiable.

D'autres systèmes de preuve par résolution existent. Certains plus puissants comme la résolution étendue [TSEITIN 1968] et d'autres plus faibles comme la résolution unitaire. Une preuve par *résolution unitaire* est une preuve par résolution où l'une des clauses c_k ou c_j (point 2 ci-dessus) est une clause unitaire $\{x\}$. Nous rappelons au passage que la résolution unitaire est complète pour les clauses de Horn.

Malgré sa simplicité, la résolution est difficile à mettre en œuvre efficacement. Elle nécessite souvent une complexité en espace élevée. De nombreuses restrictions basées sur la structure de la preuve par résolution ont été proposées. Elles sont généralement plus faciles à mettre en œuvre comme la résolution Tree-Like, linéaire, régulière, ou encore la résolution à la Davis Putnam (DP [DAVIS & H. 1960]). Par exemple, la résolution Tree-like utilise les clauses dérivées exactement une fois dans la preuve et elle est

Algorithm 1: Algorithme DP

Input: une CNF $\mathcal{F}$;
Output: $vrai$ si $\mathcal{F}$ est satisfiable ; $faux$ sinon

```
1  while (V_F ≠ ∅) do
2      x = ChoisirVariable(V_F);
3      V_F = V_F − {x};
4      F ⟵ F − (F_x ∪ F_¬x);
5      F ⟵ F ∪ η[x, F_x, F_¬x];
6      if (∅ ∈ F) then
7          return faux ;
8      end
9  end
10 return vrai
```

équivalente à une procédure DPLL [DAVIS *et al.* 1962] optimale. La résolution linéaire exige que chaque clause c_i dans une dérivation par résolution $\pi = [c_1, c_2, \ldots, c_k]$ est soit une clause de la formule initiale, soit une résolvante entre c_{i-1} et c_j avec $j < i - 1$.

Procédure DP

L'algorithme 1 proposé par Davis et Putnam [DAVIS & H. 1960] est l'une des premières méthodes de résolution dédiées au problème SAT. Son principe est basé sur l'application de la résolution comme principe de l'élimination des variables En effet, étant donnée une variable x, soit F_x (respectivement $F_{\neg x}$) l'ensemble de clauses de $\mathcal{F}$ contenant le littéral x (respectivement $\neg x$). La procédure *DP* consiste à générer toutes les résolvantes possibles entre F_x et $F_{\neg x}$ noté $\eta(x, F_x, F_{\neg x})$. Les deux sous-ensembles F_x et $F_{\neg x}$ sont ensuite omis de $\mathcal{F}$ et remplacés par $\eta(x, F_x, F_{\neg x})$. On note donc, qu'à chaque étape, le sous-problème généré contient une variable en moins, mais un nombre quadratique de clauses supplémentaires.

Cette méthode est connue pour être complète et permet de répondre à la question de la satisfiabilité ou à l'insatisfiabilité d'un problème SAT sous forme CNF, suivant qu'on a produit durant ce processus une clause vide ou pas. Cependant l'inconvénient majeur de cette méthode est son aspect exponentiel dû au nombre de résolvantes produites qui peut exploser. En pratique, cette technique dans sa forme originale est rarement utilisée [RISH & DECHTER 2000, CHATALIC & SIMON 2001]. Cependant, une forme limitée de *DP* est à la base d'un des meilleurs pré-traitements de formules CNF, la méthode "SateELite" [EÉN & BIERE 2005], intégrée dans la plupart des solveurs SAT modernes. Cette forme limitée applique la résolution pour éliminer une variable uniquement si la taille de la formule n'augmente pas. Ce qui est le cas pour un nombre non négligeable de variables dans le cas des instances issues d'applications réelles.

Exemple 2 *Soit $\mathcal{F}$ une formule CNF définie comme suit :*

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

L'élimination de la variable x_1 par la procédure DP décrite précédemment, nous permet d'obtenir la formule $\mathcal{F}'$ suivante :

$$\mathcal{F}' = \begin{cases} (c'_1) & (x_2 \vee x_3) \\ (c'_2) & (x_2 \vee \neg x \vee y) \\ (c'_3) & (x_2 \vee x \vee z) \\ (c'_4) & (x_2 \vee \neg x \vee z) \\ (c'_5) & (x_2 \vee x \vee \neg z) \\ (c'_6) & (x_2 \vee \neg y \vee \neg z) \end{cases}$$

Énumération

L'idée de base des algorithmes énumératifs (ou de retour-arrière chronologique) est la construction d'un arbre binaire de recherche où chaque nœud représente une sous-formule simplifiée par l'instanciation partielle courante. À chaque nœud de l'arbre, une variable de décision est choisie et affectée à une valeur de vérité $vrai$ ou $faux$. Le choix de la prochaine variable à instancier fait l'objet en général d'une heuristique. Deux cas peuvent se présenter, soit on a affecté toutes les variables sans contradiction (pas de clause fausse) et la formule est satisfiable, soit on a rencontré une contradiction (clause vide), et dans ce cas un retour-arrière chronologique est effectué (remise en cause de la valeur de vérité de la dernière variable de décision). La formule est déclarée insatisfiable si ce retour-arrière est effectué au-delà de la première décision. Ce schéma d'énumération est généralement appelé algorithme de Quine.

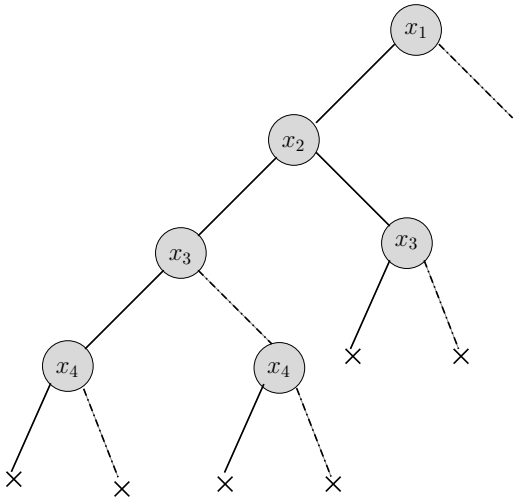


FIG. 1.1 – Retour arrière chronologique

La figure 1.1 schématise l'algorithme énumératif de Quine. On commence donc par affecter x_1 à vrai puis on explore le sous-espace à partir de ce nœud jusqu'à ce qu'on réfute cette décision. Ceci montre donc que l'affectation $x_1 = vrai$ ne fait pas partie d'une solution du problème. En d'autres termes $\mathcal{F}|_{x_1}$ est inconsistant. Notons, cependant, que l'algorithme de Quine n'est pas capable d'atteindre une contradiction, jusqu'à ce qu'il affecte aussi les variables x_2, x_3 et x_4. Cette algorithme n'utilise aucune forme de déduction ou d'inférence. Lorsque l'algorithme de Quine détecte une contradiction avec l'interprétation $\{x_1, x_2, x_3, x_4\}$ il effectue un retour-arrière et affecte $\neg x_4$ qui, elle aussi mène vers une contradiction. On effectue un retour-arrière chronologique au niveau de x_3 et on teste $\neg x_3$ et ainsi de suite jusqu'à réfuter la décision $x_1 = vrai$.

Propagation Unitaire

La propagation unitaire (PU), aussi connue sous le nom de propagation de contraintes booléennes (BCP), est l'équivalent sémantique de la résolution unitaire. La PU est l'un des procédés clés dans les algorithmes de résolution de SAT et qui est sans conteste la plus utilisée. Son principe de fonctionnement est le suivant : tant que la formule contient une clause unitaire, affecter son littéral à $vrai$.

Formellement, soit $\mathcal{F}|_x = \{c \mid c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \mathcal{F}, \neg x \in c\}$, la formule obtenue en éliminant les clauses contenant x et en supprimant $\neg x$ des clauses le contenant. Cette notation est étendue aux interprétations : soit $\rho = \{x_1, \ldots, x_n\}$ une interprétation, on définit $\mathcal{F}|_\rho = (\ldots((\mathcal{F}|_{x_1})|_{x_2})\ldots|_{x_n})$.

Définition 20 (propagation unitaire) *On note $\mathcal{F}^*$ la formule obtenue à partir $\mathcal{F}$ par application de la propagation unitaire. $\mathcal{F}^*$ est définie récursivement comme suit : (1) $\mathcal{F}^* = \mathcal{F}$ si $\mathcal{F}$ ne contient aucune clause unitaire. (2) $\mathcal{F}^* = \bot$ si $\mathcal{F}$ contient deux clauses unitaires $\{x\}$ et $\{\neg x\}$, (3) autrement, $\mathcal{F}^* = (\mathcal{F}|_x)^*$ tel que x est le littéral qui apparaît dans une clause unitaire de $\mathcal{F}$.*

Définition 21 (déduction par propagation unitaire) *Soit $\mathcal{F}$ une CNF et $x \in \mathcal{V}(\mathcal{F})$. On dit que x est déduit par propagation unitaire de $\mathcal{F}$, notée $\mathcal{F} \models_* x$, si et seulement si $(\mathcal{F} \wedge \neg x)^* = \bot$. Une clause c est déductible par propagation unitaire de $\mathcal{F}$, si et seulement si $\mathcal{F} \wedge \overline{c} \models_* \bot$ i.e, $(\mathcal{F} \wedge \overline{c})^* = \bot$*

Définition 22 (littéral pur) *Soit $\mathcal{F}$ une CNF et $x \in \mathcal{L}(\mathcal{F})$. x est appelé litéral unitaire de $\mathcal{F}$ si $\neg x$ n'apparait pas dans les clauses de $\mathcal{F}$*

Propriété 2 *Soit $\mathcal{F}$ une CNF et $x \in \mathcal{L}(\mathcal{F})$.*
un litéral pur x peut être propagé à $vrai$ en préservant la validité.

Procédure DPLL

Dans cette section, nous présentons la procédure de Davis, Putnam, Logemann et Loveland (communément appelée DPLL) [DAVIS *et al.* 1962]. L'algorithme DPLL est basé sur une recherche systématique de l'espace de recherche. IL peut être vu comme l'algorithme de Quine avec une étape supplémentaire d'inférence à chaque nœud de l'arbre : la propagation des littéraux purs et des mono-littéraux. À chaque nœud, l'interprétation partielle courante est étendue par un littéral de décision et par une séquence de littéraux propagés. Le choix de la prochaine variable à affecter est fait suivant une heuristique (e.g. choix de la variable figurant le plus souvent dans les clauses les plus courtes). La différence entre les deux procédures DP et DPLL réside dans le fait que la première procède par élimination de variables en remplaçant le problème initial par un sous-problème plus large, alors que la seconde procède par séparation. Pour une formule $\mathcal{F}$ contenant la variable x, on a $\mathcal{F}$ est satisfiable ssi $\mathcal{F}|_x$ ou $\mathcal{F}|_{\neg x}$ est satisfiable. DPLL est une méthode de type "choix+propagation".

Exemple 3 *Soit la formule CNF suivante :*

$$\mathcal{F} = \begin{cases} (c_1) & (\neg x_1 \vee x_2) & \wedge \\ (c_2) & (\neg x_2 \vee x_3) & \wedge \\ (c_3) & (\neg x_3 \vee x_4) & \wedge \\ (c_4) & (\neg x_1 \vee \neg x_4) \end{cases}$$

La figure 1.2 illustre l'importance de la propagation unitaire dans une procédure DPLL. En effet, dans le cas de DPLL (figure de droite) une décision x_1 suivie de la propagation unitaire est suffisante pour réfuter la décision d'affecter x_1 à $vrai$. Dans le cas de l'algorithme de Quine (figure de gauche), un sous-arbre a été développé avant de réfuter une telle décision.

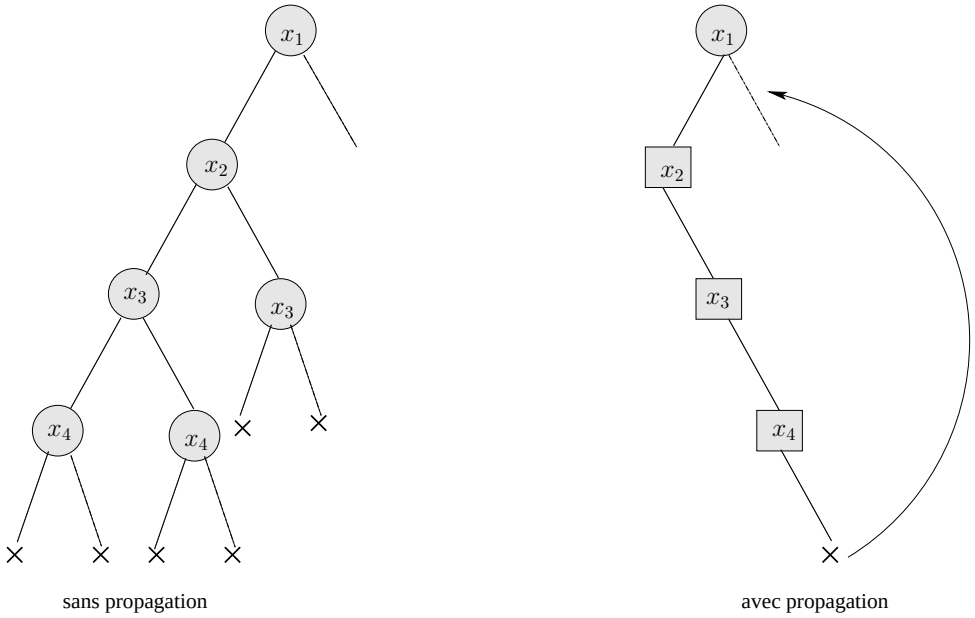


FIG. 1.2 – Quine vs DPLL

Par sa simplicité, la procédure DPLL est l'une des plus utilisées pour résoudre SAT et est actuellement la plus performante en pratique. Les deux points clés de cette procédure sont la simplification et l'heuristique de branchement utilisée. La première permet de transformer la formule en une formule plus simple équivalente pour SAT ; la seconde a une incidence directe sur la taille de l'arbre de recherche. De nombreuses améliorations lui ont été apportées, elles concernent généralement un (ou plusieurs) des points suivants :

1. choix de la prochaine variable à affecter ;
2. simplification de la formule ;
3. traitement des échecs.

Les améliorations à l'origine des solveurs SAT modernes seront présentées dans la section 1.2.3

Heuristiques de Choix de Variables

Du fait de la relation forte entre l'ordre d'affectation des variables et la taille de l'arbre de recherche développé, une « bonne » heuristique de branchement est déterminante pour l'efficacité d'un algorithme de recherche. De nombreuses heuristiques ont été proposées dans la littérature, jusqu'à récemment, la plupart d'entre elles sont définies par des arguments syntaxiques comme la longueur des clauses, le nombre d'occurrences, etc. Ces heuristiques ont cessé d'être utilisées depuis l'apparition des solveurs SAT modernes. Nous donnons ci-dessous (liste non exhaustive) quelques unes de ces heuristiques.

Heuristique MOMS L'heuristique MOMS (Maximum Occurrences in Clauses of Minimum Size) [DAVIS *et al.* 1962] est l'une des heuristiques les plus simples. Elle sélectionne la variable ayant le plus d'occurrences dans les clauses les plus courtes. Ce choix se justifie par le fait qu'elle favorise la propagation des littéraux unitaires.

Heuristique JW L'heuristique Jeroslow-Wang (JW) [JEROSLOW & WANG 1990] est basée également sur la longueur des clauses. Elle donne aussi plus de poids aux variables apparaissant dans les clauses

Algorithm 2: Algorithme DPLL

```
1  Fonction DPLL(F);
2  if (∃c ∈ F | c = (u)) ;                    // propagations des clauses unitaires
3  then
4      return DPLL(F|_u);
5  end
6  if (∃c ∈ F | c = ⊥) ;                                          // Réfutation
7  then
8      return faux
9  end
10 x ⟵ variable non affecté de F;
11 if DPLL(F|_x) = vrai ;                                          // Division
12 then
13     return vrai ;                                          // Satisfiabilité
14 end
15 else
16     return DPLL(F|_¬x) ;
17 end
```

les plus courtes. Contrairement à l'heuristique MOMS, l'heuristique *JW* tend à l'équilibrage du poids du littéral positif et négatif.

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

où $|c|$ est la taille de la clause c et α une constante. La variable x sélectionnée par l'heuristique *JW* est celle qui maximise $JW(x)$

Heuristique UP Les heuristiques *MOMS* et *JW* sélectionnent une variable en fonction de l'état courant du processus de recherche. Il semble encore plus intéressant de sélectionner une variable en prévoyant à l'avance son influence sur le processus de la recherche. L'heuristique *UP* (pour Unit Propagation) [LI & ANBULAGAN 1997] est basée sur ce principe et prédit l'évolution de la formule si une variable particulière est sélectionnée. Cette étude de l'influence de la variable est calculée via la procédure de propagation unitaire. Pour une variable non affectée x de la formule $\mathcal{F}$ étudiée, $(\mathcal{F}|_x)^*$ et $(\mathcal{F}|_{\neg x})^*$ sont calculées indépendamment permettant ainsi d'estimer le poids de la variable x par le calcul de la réduction de la taille de la formule sur les deux branches. Ce traitement permet aussi de déduire certains littéraux impliqués ou de détecter des inconsistances locales. Cette méthode fonctionne surtout sur les problèmes aléatoires là où l'apprentissage porte peu et a été intégrée sous diverses formes dans les solveurs basés sur le "lookahead" [LI & ANBULAGAN 1997, DEQUEN & DUBOIS 2006, ZHANG *et al.* 2001] en choisissant par exemple un sous-ensemble de variables à tester (par exemple les littéraux apparaissant dans des clauses binaires) [LI & ANBULAGAN 1997].

Simplifications

Une technique de simplification est intéressante si elle permet non seulement de réduire le nombre de nœuds de l'arbre de recherche, mais également le temps de calcul. En plus de la propagation des littéraux

purs et unitaires, de nombreuses techniques de simplification ont été introduites et utilisées, soit comme pré-traitement à la formule, soit au cours de la recherche. Parmi elles, on peut citer par exemple :

- la *résolution restreinte* qui consiste à ajouter des résolvantes de taille bornée. Cette technique est très souvent utilisée comme pré-traitement à la formule et permet en particulier de détecter certaines inconsistances locales, et de mieux traiter certaines classes de formules [BOUFKHAD 1996]. On peut également citer l'hyper-résolution binaire introduite par Bacchus dans [BACCHUS 2002] et la production de sous-clauses en utilisant la propagation unitaire [DARRAS *et al.* 2005]. Finalement les deux meilleurs pré-traitements actuellement intégrés dans les solveurs SAT modernes "SatELite"[EÉN & BIERE 2005] et "ReVival"[PIETTE *et al.* 2008] se basent aussi sur l'utilisation d'une forme de résolution limitée. SatELite est basé sur une opération classique d'élimination de variables par résolution i.e, procédure DP limitée. L'élimination d'une variable est effectuée uniquement si cette opération n'augmente pas la taille de la formule [SUBBARAYAN & PRADHAN 2004]. SatELite exploite aussi des propriétés structurelles de la formule : les dépendances fonctionnelles [GREGOIRE *et al.* 2005]. ReVival est basée sur un traitement des redondances par propagation unitaire. Le test de redondance par propagation unitaire d'une clause est capturé par un graphe d'implication qui est lui même utilisé pour déduire une sous-clause.
- Les *traitements locaux par propagation unitaire* ont été introduits dans CSAT par Olivier Dubois et al. [DUBOIS *et al.* 1996]. Ces traitements, appliqués pendant la résolution, permettent de produire par propagation unitaire des littéraux impliqués ou de détecter des inconsistances locales.
- L'*exploitation des classes polynômiales* au cours de la résolution a fait l'objet de nombreux travaux. Parmi les restrictions polynômiales connues, 2-SAT et Horn SAT restent les plus exploitées dans le cadre général [GALLO & URBANI 1989, BUNO & BUNING 1993, BOROS *et al.* 1994, LARRABEE 1992]. Ceci s'explique en partie par le fait que dans de nombreux problèmes réels, certaines contraintes peuvent être codées par des clauses binaires ou des clauses de Horn.

Analyse de conflits et retour-arrière non chronologique

L'un des problèmes majeurs du retour-arrière chronologique réside dans la non prise en compte des raisons qui ont mené au conflit.

L'analyse des conflits remédie à ce problème en tenant compte des décisions à l'origine du conflit. Cette analyse permet d'effectuer un retour-arrière non chronologique, et d'éviter de redécouvrir les mêmes échecs. En effet, ce processus permet d'effectuer un retour-arrière à un niveau $m < n$ sans nécessairement essayer toutes les possibilités entre le niveau du conflit n et le niveau m.

Ce type d'approches a été étudié et appliqué dans de nombreux domaines de l'IA, comme dans les systèmes de maintien de vérité *ATMS* [MCALLESTER 1980, STALLMAN & SUSSMAN 1977], les problèmes de satisfaction de contraintes [DECHTER 1989, GINSBERG 1993, SCHIEX & VERFAILLIE 1993] et en programmation logique [BRUYNOOGHE 1980]. Elles se distinguent essentiellement par les techniques utilisées pour analyser les échecs et pour éviter de rencontrer les mêmes situations au cours de la recherche.

L'analyse du conflit permet d'identifier dans l'interprétation courante un sous-ensemble de décisions (appelé ensemble conflit) à l'origine de la dérivation de la clause vide [JR. & SCHRAG 1997]. Ce sous-ensemble permet de déterminer la dernière décision dans la branche courante à l'origine du conflit. Un retour-arrière non chronologique est ensuite effectué pour remettre en cause la valeur de vérité de cette variable.

Dans l'exemple 2 , après avoir affecté $x_1 = vrai$, $x_2 = vrai$, $x_3 = vrai$ et $x = vrai$, la propagation unitaire permet de propager $y = vrai$ (clause c_3), $z = vrai$ (clause c_5) et la clause c_7 devient vide (tous les littéraux sont à $faux$). Dans ce cas, l'ensemble conflit est l'ensemble des variables qui ont mené au conflit $\{x_1 = vrai, x = vrai, y = vrai, z = vrai\}$. Notons que x_2 et x_3 ne participent pas

au conflit. L'algorithme teste ensuite l'autre valeur de vérité de x. Supposons qu'après avoir affecté x à $faux$, l'algorithme détecte un nouveau conflit. et que l'ensemble conflit est $\{x_1 = vrai, x = vrai, z = vrai\}$. Du moment qu'on a testé de manière exhaustive les valeurs de vérité de x, le retour-arrière non chronologique peut maintenant se faire au niveau de x_1 qui est affecté à $faux$ et le processus est réitéré. Nous pouvons remarquer que, cette fois, l'algorithme est en mesure de sortir du conflit sans essayer les décisions intermédiaires x_1 ou x_2.

Le retour-arrière non chronologique permet d'éviter partiellement les mêmes conflits ou "trashing". Il peut toutefois répéter les mêmes erreurs dans le futur. On peut remédier à ce problème en ajoutant des clauses représentant le conflit ("nogood"). À chaque fois qu'une contradiction est détectée, cela présente une opportunité d'identifier une clause impliquée par la CNF permettant à la propagation unitaire de réaliser de nouvelles propagations. Cette clause est connue sous le nom de *clause apprise* (conflict-driven clause) [MARQUES-SILVA & SAKALLAH 1996, ZHANG *et al.* 2001, BEAME *et al.* 2004, JR. & SCHRAG 1997]. L'ajout de clauses à la base permet donc à la propagation unitaire de découvrir de nouvelles implications et permettra de ne pas répéter les mêmes conflits dans le futur. Dans l'exemple, après l'affectation de $x_1 = vrai$ et $x = vrai$, l'algorithme détecte une contradiction et peut donc ajouter la clause $(\neg x_1 \vee \neg x)$. Cette clause n'appartient pas à la formule originale car dans le cas contraire on aurait déduit le littéral unitaire $\neg x$ et évité ainsi le conflit.

Solveurs SAT modernes

Les progrès récents obtenus dans le cadre de la résolution pratique du problème SAT font généralement référence à l'efficacité remarquable des solveurs SAT (appelés solveurs SAT modernes). La mise en œuvre des solveurs SAT modernes est rendue possible (liste non exhaustive) grâce à une meilleure compréhension de la nature de la difficulté des problèmes (e. g. phénomène *heavy tailed* [GOMES *et al.* 1997], *backdoor* [WILLIAMS *et al.* 2003], et *backbone* [DEQUEN & DUBOIS 2006]), à l'exploitation des propriétés structurelles (e. g. dépendances fonctionnelles [GREGOIRE *et al.* 2005]), à la mise en œuvre efficace des techniques d'apprentissage à partir des échecs [MARQUES-SILVA & SAKALLAH 1996, JR. & SCHRAG 1997], à l'introduction de nouveaux paradigmes algorithmiques (e.g. la *randomization* et les *redémarrages* [GOMES *et al.* 1998]), à la mise en œuvre d'heuristiques adaptatives dirigées par les conflits [BRISOUX *et al.* 1999] et à la proposition de structures de données efficaces (e. g. *watched literals* [ZHANG 1997, ZHANG *et al.* 2001]).

Dans la suite, nous donnons une brève description des éléments essentiels d'un solveur SAT moderne. Une description plus formelle sera donnée dans le chapitre 2.

Apprentissage L'utilisation du retour-arrière non chronologique dans les solveurs modernes est couplé avec le mécanisme d'apprentissage. La combinaison de ces techniques assure la propagation unitaire d'être plus robuste à chaque apparition d'un nouveau conflit et permet au solveur de ne pas répéter les mêmes échecs. L'apprentissage des clauses à partir des conflits est effectué via un processus appelée *analyse de conflits*, qui analyse la trace de la propagation unitaire sur un graphe connu sous le nom de *graphe d'implication*. Ce graphe, connu pour être acyclique, est une représentation des clauses ayant été à l'origine de la propagation des littéraux unitaires.

La clause à apprendre est dérivée à partir du graphe d'implication notée $c_a = (\neg d \vee \alpha)$, ou α est une disjonction de littéraux. Elle est obtenue par un processus de résolution entre la clause conflit et l'ensemble des clauses impliquant les littéraux de celle-ci. La clause finale obtenue est appelée *clause assertive*. Cette dernière est ajoutée à la base des clauses. Elle présente la particularité de contenir un littéral du dernier niveau de décision appelé *littéral assertive* $\neg d$ et les autres littéraux de α ont des niveaux inférieurs au niveau courant. Le processus consiste donc à effectuer un retour-arrière au niveau maximal des littéraux de α et à propager $\neg d$ à ce niveau appelé *niveau assertive*. Le processus qui consiste

Algorithm 3: Algorithme CDCL

Input: une formule CNF $\mathcal{F}$;
Output: $vrai$ if $\mathcal{F}$ si satisfiable ; $faux$ sinon

```
1  begin
2      currentLevel = 0;
3      simplifier(F);
4      while (vrai) do
5          confl = propagate();
6          if (confl ≠ NULL) then
7              if (currentLevel==0) then return faux;
8              (c, backJumpLevel)=analyze-conflict();
9              addlearnt(c);
10             currentLevel = backJumpLevel;
11             bactrack(backJumpLevel);
12         end
13         else
14             currentLevel++;
15             v = pickvariable();
16             if (v == undef) then
17                 return vrai
18             end
19         end
20     end
21 end
```

à ajouter les clauses à la formule originale est communément connu sous le nom d'*apprentissage dirigé par les conflits* [MARQUES-SILVA & SAKALLAH 1996, ZHANG *et al.* 2001, BEAME *et al.* 2004, JR. & SCHRAG 1997, SANG *et al.* 2005].

L'algorithme 3 décrit globalement le principe de fonctionnement d'un solveur moderne. La fonction *simplifier* permet de propager les clauses unitaires. L'algorithme commence par choisir une variable et décide de la propager grâce à la fonction *propagate*, ce processus est répété jusqu'à ce que l'algorithme trouve une solution ou un conflit. En cas de conflit, l'algorithme analyse le conflit via la fonction *analyse-conflit*. Cette analyse permet d'extraire une nouvelle clause assertive qui sera ajoutée à la base de nogoods et un niveau assertif qui correspond au niveau du retour-arrière. L'algorithme effectue un saut à ce niveau et affecte grâce à la propagation unitaire le littéral assertif.

Dans [BEAME *et al.* 2004], les auteurs montrent que la recherche basée sur DPLL et qui utilise le principe d'apprentissage de clauses peut générer une preuve exponentiellement plus courte qu'une recherche ne se basant pas sur le principe l'apprentissage.

Le Redémarrage La majorité des solveurs SAT modernes ont une stratégie de redémarrage, qui consiste à démarrer une nouvelle recherche et à priori avec un nouvel ordre des variables. Cette stratégie a été proposée par [GOMES *et al.* 1998]. La plupart de ces solveurs effectuent un redémarrage après qu'un certain seuil de clauses apprises (ou de conflits) est atteint. Le parcours de l'espace de recherche change d'un redémarrage à l'autre, dû soit à l'aspect aléatoire présent dans les algorithmes de recherche, soit au changement de la formule d'entrée augmentée par les clauses apprises ou les deux à la fois.

L'argument majeur derrière l'adoption du redémarrage est prouvé expérimentalement en utilisant

plusieurs ordres de variables. Les résultats ont montré une variation énorme en terme d'efficacité suivant l'ordre considéré qui peut varier de plusieurs ordres de grandeur. Une autre observation faite dans [RYVCHIN & STRICHMAN 2008] montre qu'en général la taille des clauses apprises en haut de l'arbre de recherche est en moyenne plus petite qu'en bas de l'arbre.

Différentes politiques de redémarrage ont été proposées récemment. Un survol de la majorité de ces politiques est décrit dans [HUANG 2007]. On décrit brièvement ici les plus connues parmi elles.

1. Série arithmétique paramétrée par x, y : ce sont des stratégies où un redémarrage est effectué après chaque x conflits et qui augmente de y après chaque redémarrage. Quelques valeurs de référence : Zchaff 2004 $x = 700$, dans BerkMin $x = 550$, dans Siege $x = 16000$ et dans EUREKA [NADEL *et al.* 2006] $x = 2000$. Dans tous ces solveurs l'incrément est fixé (i.e, y = 0).
2. Série géométrique parametrée par x, y : ce sont des stratégies dans lesquelles on a un redémarrage chaque x conflits. La valeur de x est multipliée par le facteur y à chaque redémarrage. Dans MINISAT 2007 par exemple $x = 100$ et $y = 1.5$.
3. Luby [LUBY *et al.* 1993a, RYVCHIN & STRICHMAN 2008] série paramétrée par x : est une stratégie qui évolue grâce à la suite de nombres suivante 1, 1, 1, 2, 1, 1,2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1... multipliés par la constante x (appelée unité de calcul). Formellement, soit t_i le i^{eme} nombre de cette série. t_i est défini récursivement comme suit :

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

   Cette dernière stratégie a quelques caractéristiques théoriques intéressantes dans la classe des algorithmes aléatoires appelés $LasVegas$ mais son adaptation au cas SAT est récente et il n'existe pas d'explications théoriques à son succès dans les solveurs CDCL vu qu'aujourd'hui elle est devenue le choix des solveurs modernes comme Rsat [PIPATSRISAWAT & DARWICHE 2007a] ou Tinysat [HUANG 2007] ou récemment MINISAT.

Pour compléter cette famille de stratégies, il faut noter qu'il existe d'autres stratégies qui consistent à faire un redémarrage non pas au sommet de l'arbre de recherche mais à un niveau plus bas [LYNCE *et al.* 2001] dans l'arbre de recherche et qui tiennent compte du calcul de l'évolution de la recherche sur les branches de l'arbre de recherche plutôt que la taille de l'arbre lui-même.

Toutes les stratégies décrites précédemment sont donc basées sur un compteur global du nombre de clauses apprises et, par conséquent, elles mesurent les progrès réalisés au cours de la résolution du problème. En supposant que la motivation pour le redémarrage est de prévenir le solveur au cas où il se trompe de branche.

Heuristique VISDS Si les heuristiques citées précédemment ont donné quelques résultats intéressants sur des instances particulières, la majorité d'entre elles sont coûteuses et quelques unes présentent l'inconvénient d'être liées seulement aux occurrences des littéraux dans les clauses. L'évolution des heuristiques classiques a été marquée par l'apparition d'heuristiques dynamique dirigée par les conflits et dont le coût de calcul est négligeable et qui présente la particularité d'être indépendantes de l'interprétation courante.

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

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

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

En 2001 dans [ZHANG *et al.* 2001], les auteurs proposent une heuristique appelée VSIDS (Variable State Independent Decaying Sum). Cette heuristique associe à chaque variable un compteur (activité). A chaque fois qu'une clause est violée, toutes les variables de cette clause voient leurs compteurs incrémentés d'un facteur additionnel. A chaque décision, la variable accumulant la meilleure activité est choisie. Cette heuristique dont le coût est dérisoire est facile à mettre à jour est dirigée par les conflits. Les variables apparaissant le plus dans les conflits sont jugées les plus pertinentes. Une autre généralisation de cette heuristique a été proposée dans le solveur BerkMin [GOLDBERG & NOVIKOV 2002]. Le principe est d'augmenter l'activité de toutes les variables rencontrées lors de la génération de la clause assertive. Cette dernière est la plus utilisée actuellement. Reste qu'associer un compteur à chaque variable ne permet pas de décider de la valeur de vérité à choisir lors d'une décision. Pour pallier ce problème, dans [PIPATSRISAWAT & DARWICHE 2007a] les auteurs proposent d'enregistrer la valeur de vérité affectée à toutes les variables lors du retour-arrière ou du redémarrage. A chaque nouvelle décision, la variable choisie est affectée à sa dernière valeur de vérité enregistrée. Ce mécanisme déjà exploité dans le cadre du problème de satisfaction de contrainte (CSPs) a montré son efficacité sur plusieurs classes d'instances de problèmes SAT.

Le mécanisme d'apprentissage permet de stocker des clauses permettant d'éviter les conflits déjà enregistrés. Cependant avec la progression de la recherche, la taille de la formule croît rapidement et la pertinence de toutes les clauses apprises est mise en cause. Dans la majorité des solveurs modernes un mécanisme de réduction de la base est mis en place. Il consiste à supprimer une partie des clauses apprises de temps à autre. Reste à savoir quelles clauses faut il supprimer. Un lien direct a été trouvé entre l'heuristique VSIDS et cette opération. Comme pour les variables, celle-ci consiste à associer à chaque clause un compteur (activité). A chaque conflit, toutes les clauses apprises participant à la construction de la clause assertive voient leurs compteurs incrémentés. Lors de la réduction de la base de clauses apprises, les clauses ayant les activités les plus faibles sont supprimées. Cependant, aucune étude n'a montré jusqu'à maintenant le pourcentage optimal de clauses à supprimer. Dans MINISAT [EÉN & SÖRENSSON 2003] par exemple, au moins la moitié des clauses est supprimée lorsque la base de nogoods est jugée trop excessive en taille par rapport à la formule originale.

Les structures de données paresseuses ("Watched literals") Pour capturer la propagation unitaire, auparavant les structures de données étaient moins robustes. Dû aux grandes avancées dans le domaine des heuristiques de branchement et l'apprentissage, il est devenu indispensable de disposer de structures de données moins coûteuses, plus robustes et capables de capturer celle-ci avec un coût minimal. Dans les solveurs modernes, 80% du temps de calcul est consommé par la propagation unitaire. Ce qui rendait l'optimisation de ces structures primordiale. La grande avancée dans ce domaine a été proposée par [ZHANG *et al.* 2001]. En effet, la constatation est d'autant plus évidente qu'un littéral ne peut être propagé tant qu'au moins deux littéraux de cette clause ne sont pas affectés. Les auteurs ont proposé donc de garder deux pointeurs sur deux littéraux de chaque clause, appelés *watched literals* (voir figure 1.3). A chaque fois que l'un de ces littéraux est affecté à $faux$ le pointeur est déplacé vers un autre littéral qui n'est pas $faux$. Lorsque les deux littéraux pointent le même littéral, cela implique que ce littéral est unitaire et pourra être propagé. L'intérêt de ces structures est double, elles permettent d'un côté de capturer la propagation unitaire et de l'autre de ne visiter que les clauses concernées par les affectations

actuelles. Dans *Picosat*, l'auteur propose de fixer les deux pointeurs sur les deux premiers littéraux de chaque clause. A chaque fois qu'un de ces deux littéraux devient faux, au lieu de déplacer le pointeur, le solveur cherche dans le reste de la clause un autre littéral non affecté à $faux$ et effectue une permutation. L'un des avantages majeurs de cette structure se situe en sa capacité à ne parcourir, lors d'une affectation d'un littéral x, que les clauses dans lesquelles $\neg x$ est pointé. Il est également important de noter qu'aucune mise à jour n'est nécessaire lors du retour-arrière.

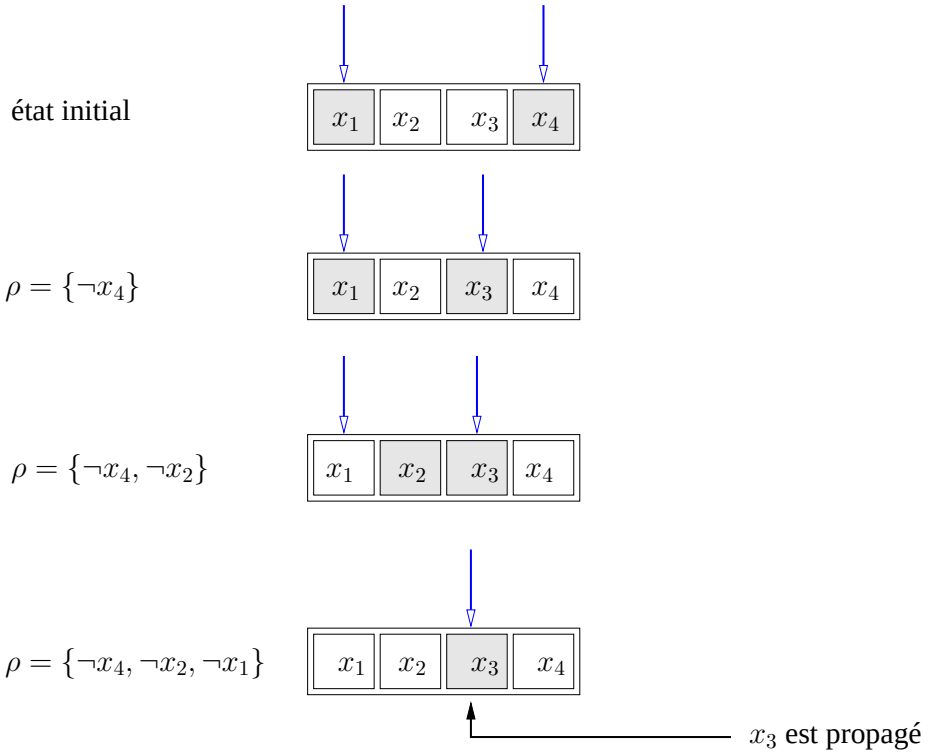


FIG. 1.3 – watched literals

1.3 Problèmes autour de SAT

Le nombre de problèmes qui gravitent autour de SAT accentue l'importance du problème SAT et élargit ses champs d'applications. Même si ces problèmes sont généralement beaucoup plus difficiles que SAT, ils bénéficient régulièrement de transferts ou d'extensions de résultats obtenus dans le cadre SAT.

Nous rappelons certains des problèmes qui suscitent un intérêt croissant ces dernières années.

Définition 23 (MaxSAT) *`MaxSAT` est le problème d'optimisation associé à SAT. Il s'agit, pour une formule CNF donnée $\mathcal{F}$ de déterminer le nombre maximal MaxSAT($\mathcal{F}$) de clauses de $\mathcal{F}$ qui peuvent être satisfaites.*

Remarque 2 *Pour une formule CNF $\mathcal{F}$ satisfiable composée de n clauses, on a MaxSAT($\mathcal{F}$) = n.*

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

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

Définition 24 (#SAT) *Soit $\mathcal{F}$ une formule CNF. #SAT est le problème qui consiste à déterminer le nombre de modèles de $\mathcal{F}$.*

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

2

Un cadre étendu pour l'analyse de conflits

Sommaire

2.1 Introduction 18
2.2 Quelques notations préliminaires 19
2.3 Analyse des conflits : approche classique (CDCL) 20
2.3.1 Graphe d'implications 20
2.3.2 Génération des clauses assertives 21
2.4 Preuve d'optimalité du schéma "First UIP" 23
2.5 eCDCL : un cadre étendu pour l'analyse de conflits 24
2.5.1 Graphe d'implications étendu 24
2.5.2 Génération de clauses assertives étendues 26
2.6 Mise en œuvre en pratique 28
2.6.1 Améliorer la qualité du saut arrière 28
2.6.2 Réduire la taille des clauses assertives 28
2.7 Expérimentations 29
2.8 Conclusion 30

Ce chapitre présente plusieurs contributions au schéma CDCL ("*Conflict Driven Clauses Learning*"), une des composantes clés des solveurs SAT modernes (voir section 2.3). Tout d'abord, nous montrons que, à partir du graphe d'implication, les clauses assertives obtenues en utilisant le principe du premier point d'implication unique "*First Unique Implication Point*" (FUIP) sont optimales en terme de retours-arrière. Puis nous proposons une extension [AUDEMARD *et al.* 2008a] du graphe d'implication contenant des arcs additionnels appelés arcs inverses. Ces arcs sont obtenus en tenant compte des clauses satisfaites qui sont habituellement ignorées par l'analyse du conflit. Cette extension capture plus fidèlement l'ensemble du processus de propagation et ouvre de nouvelles perspectives pour les approches fondées sur CDCL. Entre autres avantages, notre extension du graphe d'implication conduit à un nouveau schéma d'analyse des conflits qui exploite les arcs ajoutés et permet des retours-arrière plus haut dans l'arbre de recherche. Les résultats expérimentaux montrent que l'intégration de notre système d'analyse des conflits généralisé au sein de solveurs les plus efficaces améliore sensiblement leurs performances.

2.1 Introduction

La structure de données principale d'un solveur basé sur CDCL est le graphe d'implication qui enregistre les différentes propagations faites durant la construction de l'interprétation partielle. Ce graphe

d'implication permet d'analyser les conflits. Il est utilisé pour effectuer du retour-arrière ("*backtracking*") intelligent, pour apprendre des nogoods et pour mettre à jour les poids des variables dans les heuristiques dynamiques de type VSIDS [MOSKEWICZ *et al.* 2001, GOLDBERG & NOVIKOV 2002]. Il est important de noter que ce graphe est construit de manière incomplète, il ne donne qu'une vue partielle des implications entre les littéraux. Regardons par exemple ce qui se passe lorsque un littéral y est déduit à un niveau donné. Cette déduction est faite parce qu'une clause, par exemple $(\neg x_1 \vee \neg x_2 \vee y)$, est devenue unitaire sous l'interprétation courante. Cette dernière contient donc les affectations $x_1 = vrai$ et $x_2 = vrai$ qui ont eu lieu à des niveaux inférieurs ou égaux au niveau de l'affectation de y. Lorsque y est impliqué, la raison de cette implication est enregistrée, on ajoute donc au graphe d'implication des arcs entre x_1 et y et entre x_2 et y. Supposons maintenant que $(x_1 \vee \neg y)$ soit une autre clause de la formule, x_1 est donc une conséquence de l'affectation de y. Il serait correct d'ajouter l'arc (y, x_1) au graphe d'implication, mais cela n'est pas le cas. En effet, seule la première explication (clause) rencontrée d'un littéral déduit est enregistrée dans le graphe. Cette stratégie est donc fortement dépendante de l'ordre des clauses dans la formule. Nous proposons une extension du graphe d'implication dans lequel un littéral déduit peut avoir plusieurs explications. Contrairement au cas traditionnel, les arcs peuvent aller en arrière par rapport aux niveaux d'affectations, c'est le cas par exemple si nous rajoutons l'arc (y, x_1) puisque y a un niveau supérieur (il est propagé après) celui de x_1. Ces arcs particuliers sont appelés *arcs inverses*.

Nous proposons donc une extension forte du graphe d'implication, basée sur la notion de graphe d'implication généralisé que nous venons de présenter de manière informelle. Nous commençons par prouver que pour un graphe d'implication donné, la clause conflit assertive générée en utilisant le principe du premier UIP (*Unique implication point*) est optimale en terme de sauts. Nous formalisons ensuite la notion de graphe d'implication étendu et étudions son utilisation lors de l'analyse de conflits. Les arcs arrière permettent de détecter des raisons pour des variables de décision, ce qui ne peut être fait avec un graphe d'implications classique. Cela ouvre de nouvelles perspectives pour les solveurs de type CDCL. Les arcs inverses peuvent servir à minimiser les clauses assertives déduites du graphe d'implication classique ou à améliorer la hauteur du retour-arrière i.e, du saut. Ces arcs peuvent également être utilisés comme raisons alternatives lors de la génération de la clause assertive.

Le reste du chapitre est organisé comme suit. Après quelques définitions et notations utiles pour la suite, le graphe d'implication classique et son utilisation pour générer des nogoods sont formellement présentés. Nous prouvons ensuite l'optimalité en termes de saut de la clause assertive générée. Ce résultat d'optimalité explique définitivement l'intérêt d'utiliser le premier UIP. Il nous conforte également dans l'idée que pour améliorer ce schéma d'analyse des conflits, on a besoin d'étendre le graphe d'implications. Nous proposons ensuite le graphe d'implication obtenu grâce à l'ajout de nouvelles clauses (implications) issues de la partie satisfiable de la formule. Nous prouvons qu'une telle extension peut nous aider à améliorer la hauteur des sauts (des retours-arrière) ou à réduire la clause assertive. Avant de conclure ce chapitre, nous proposons une première intégration de notre approche aux solveurs de type CDCL comme Rsat [PIPATSRISAWAT & DARWICHE 2007b] ou Minisat [EÉN & SÖRENSSON 2003]. Cette nouvelle approche est validée expérimentalement sur les instances SAT industrielles issues des récentes compétitions SAT.

2.2 Quelques notations préliminaires

Nous introduisons maintenant des notations et remarques terminologiques associées aux solveurs SAT basés sur la procédure de Davis Logemann Loveland, communément appelée DPLL [DAVIS *et al.* 1962]. DPLL est une procédure de recherche de type "*backtrack*". À chaque nœud les littéraux affectés (le littéral de décision et les littéraux propagés) sont étiquetés avec le même *niveau de décision*, initialisé à 1 et incrémenté à chaque nouveau point de décision. Le niveau de décision courant est le niveau le plus

élevé dans la pile de propagation. Lors d'un retour-arrière ("*backtrack*"), les variables ayant un niveau supérieur au niveau du backtrack sont désaffectées et le niveau de décision courant est décrémenté en conséquence (égal au niveau du backtrack). Au niveau i, l'interprétation partielle courante ρ peut être représentée comme une séquence décision–propagations de la forme $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ telle que le premier littéral x_k^i correspond au littéral de décision x_k affecté au niveau i et chaque $x_{k_j}^i$ de l'ensemble $1 \leq j \leq n_k$ représente les littéraux unitaires propagés à ce même niveau i.

Soit $x \in \rho$, on note $l(x)$ le niveau d'affectation de x, $d(\rho, i) = x$ si x est le littéral de décision affecté au niveau i. Pour un niveau donné i, ρ^i représente la projection de ρ aux littéraux affectés au niveau $\leq i$.

2.3 Analyse des conflits : approche classique (CDCL)

2.3.1 Graphe d'implications

Le graphe d'implication est une représentation capturant les variables affectées durant la recherche, que ce soit des variables de décision ou des variables propagées. Cette représentation est utile pour analyser les conflits. A chaque fois qu'un littéral y est propagé, on garde en mémoire la clause à l'origine de cette propagation, clause que l'on note $\overrightarrow{cla}(y)$. La clause $\overrightarrow{cla}(y)$ est donc de la forme $(x_1 \vee \cdots \vee x_n \vee y)$ où chaque littéral x_i est faux sous l'interprétation courante ($\forall i \in 1 \ldots n \quad \rho(x_i) = faux$) et $\rho(y) = vrai$. Quand un littéral y n'est pas obtenu par propagation unitaire mais qu'il correspond à un point de choix, $\overrightarrow{cla}(y)$ est indéfini, que l'on note par convention $\overrightarrow{cla}(y) = \bot$.

Lorsque $\overrightarrow{cla}(y) \neq \bot$, on note par $exp(y)$ l'ensemble $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, appelé l'ensemble des *explications* de y. Autrement dit, si $\overrightarrow{cla}(y) = (x_1 \vee \cdots \vee x_n \vee y)$ alors les explications sont les littéraux $\overline{x_i}$ qui constituent la condition sous laquelle $\overrightarrow{cla}(y)$ devient une clause unitaire $\{y\}$. Notons que pour tout i on a $l(\overline{x_i}) \leq l(y)$, i.e, toutes les explications de la déduction ont un niveau inférieur à celui de y.

Quand $\overrightarrow{cla}(y)$ est indéfini, $exp(y)$ est égal à l'ensemble vide. Les explications peuvent, alternativement, être vues comme un graphe d'implication dans lequel l'ensemble des prédécesseurs d'un nœud correspond aux explications du littéral correspondant :

Définition 25 (Graphe d'implication) *Soient $\mathcal{F}$ une formule CNF et ρ une interprétation partielle. Le graphe d'implication associé à $\mathcal{F}$, ρ et exp est $(\mathcal{N}, \mathcal{E})$ tel que :*

- *$\mathcal{N} = \rho$, i.e, il existe exactement un seul nœud pour chaque littéral, de décision ou impliqué ;*
- *$\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in exp(y)\}$*

Exemple 4 *On considère une formule $\mathcal{F}$ contenant, entre autres, les clauses suivantes ($\mathcal{F} \supseteq \{c_1, \ldots, c_9\}$) :*

$$
\begin{array}{llll}
(c_1) & x_6 \vee \neg x_{11} \vee \neg x_{12} & (c_2) & \neg x_{11} \vee x_{13} \vee x_{16} \\
(c_3) & x_{12} \vee \neg x_{16} \vee \neg x_2 & (c_4) & \neg x_4 \vee x_2 \vee \neg x_{10} \\
(c_5) & \neg x_8 \vee x_{10} \vee x_1 & (c_6) & x_{10} \vee x_3 \\
(c_7) & x_{10} \vee \neg x_5 & (c_9) & \neg x_3 \vee \neg x_{19} \vee \neg x_{18} \\
(c_8) & x_{17} \vee \neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18} & &
\end{array}
$$

Soit ρ l'interprétation partielle suivante $\rho = \{\langle \ldots \neg x_6^1 \ldots \neg x_{17}^1 \rangle \langle (x_8^2) \ldots \neg x_{13}^2 \ldots \rangle \langle (x_4^3) \ldots x_{19}^3 \ldots \rangle \ldots \langle (x_{11}^5) \ldots \rangle\}$. Le niveau de décision courant est 5. Le graphe d'implication $\mathcal{G}_{\mathcal{F}}^{\rho}$ associé à $\mathcal{F}$ et ρ est schématisé dans la figure 2.1 (notez que seule une partie du graphe est affichée).

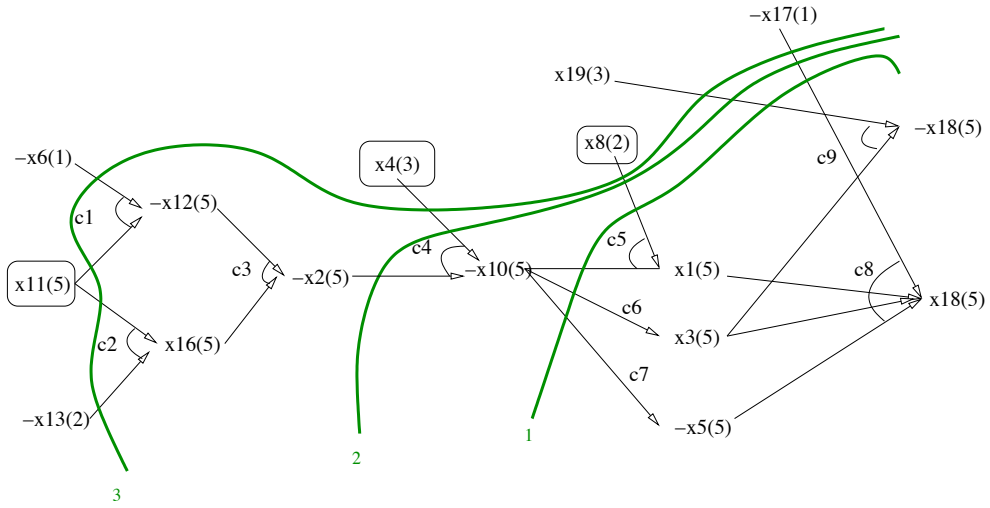


FIG. 2.1 – Graphe d'implication $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$

2.3.2 Génération des clauses assertives

Dans cette section, on décrit formellement le schéma d'apprentissage classique utilisé dans les solveurs SAT modernes. Dans les définitions suivantes, on considère $\mathcal{F}$ une formule CNF, ρ une interprétation partielle telle que $(\mathcal{F}|_\rho)* = \bot$ et $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ le graphe d'implication associé. Supposons que le niveau de décision courant est m, puisque l'on a atteint le conflit, il existe donc un littéral z tel que $\{z, \neg z\} \subset \mathcal{N}$ et $l(z) = m$ ou $l(\neg z) = m$. L'analyse du conflit est basée sur l'application de la résolution (par z) sur la clause devenue fausse et en remontant dans le graphe d'implication, et ceci en utilisant les clauses $(\overline{exp(y)} \vee y)$ à chaque nœud $y \in \mathcal{N}$. Le processus s'arrête lorsqu'il ne reste plus qu'un littéral x du dernier niveau d'affectation. On appelle ce processus la preuve par résolution basée sur les conflits. Définissons formellement les différents concepts de clause assertive, de preuve par résolution basée sur les conflits et de point d'implication unique (*Unique Implication Point*).

Définition 26 (Clause assertive) *Une clause c de la forme $(\alpha \vee x)$ est dite une clause assertive ssi $\rho(c) = faux$, $l(x) = m$ et $\forall y \in \alpha, l(y) < l(x)$. x est appelé littéral assertif, qu'on notera $\mathcal{A}(c)$. On définit également $jump(c) = max\{l(\neg y) \mid y \in \alpha\}$.*

Définition 27 (Preuve par résolution basée sur les conflits) *Une preuve par résolution basée sur les conflits π est une séquence de clause $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ qui satisfait les conditions suivantes :*

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

Notons que chaque σ_i est une résolvante de la formule $\mathcal{F}$: par induction, σ_1 est la résolvante entre deux clauses qui appartiennent à $\mathcal{F}$; pour chaque $i > 1$, σ_i est une résolvante entre σ_{i-1} (qui, par hypothèse d'induction, est une résolvante) et une clause de $\mathcal{F}$. Chaque σ_i est aussi un *impliquant* de $\mathcal{F}$, c'est-à-dire : $\mathcal{F} \models \sigma_i$. Une autre remarque importante est que dans les solveurs SAT modernes, les littéraux y utilisés dans la condition 2 de la définition 27 sont restreints à ceux du niveau courant.

Définition 28 (Preuve élémentaire) *Une preuve par résolution basée sur les conflits $\pi = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ est dite élémentaire ssi $\nexists i < k$ tel que $\langle \sigma_1, \sigma_2, \ldots \sigma_i \rangle$ soit aussi une preuve par résolution basée sur les conflits.*

En utilisant les définitions 27 et 28, on peut désormais définir les concepts du point d'implication unique (*Unique Point Implication* (UIP)) et du premier UIP :

Définition 29 (Point d'Implication Unique (UIP)) *Un nœud $x \in \mathcal{N}$ est un UIP ssi il existe une preuve par résolution basée sur les conflits $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ telle que $\overline{x} \in \sigma_k$ et $l(x)$ est égal au niveau de décision courant, m (Notez que σ_k est assertive, et a exactement un seul x de la sorte).*

Définition 30 (Premier UIP) *Un nœud $x \in \mathcal{N}$ est un premier UIP ssi il est obtenu à partir d'une preuve* élémentaire *; i.e, $\exists \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ une preuve par résolution basée sur les conflits tq. $\overline{x} \in \sigma_k$ et $l(x) = m$.*

Définition 31 (Dernier UIP) *Le dernier UIP est défini comme étant le littéral $d(\rho, m)$, i.e, le littéral de décision au niveau du conflit m.*

Pour illustrer les définitions précédentes, considérons à nouveau l'exemple 4.

Le parcours du graphe $\mathcal{G}^{\rho}_{\mathcal{F}}$ nous permet de générer trois clauses assertives correspondant aux trois UIPs possibles (voir figure 2.1). Illustrons la preuve par résolution basée sur les conflits qui conduit à la première clause assertive Δ_1 correspondant au premier UIP (voir coupe 1 de la figure 2.1).

- $\sigma_1 = \eta[x_{18}, c_8, c_9] = (x^1_{17} \vee \neg x^5_1 \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19})$
- $\sigma_2 = \eta[x_1, \sigma_1, c_5] = (x^1_{17} \vee \neg x^5_3 \vee x^5_5 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_3 = \eta[x_5, \sigma_2, c_7] = (x^1_{17} \vee \neg x^5_3 \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$
- $\sigma_4 = \Delta_1 = \eta[x_3, \sigma_3, c_6] = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee x^5_{10})$

Comme on peut le voir, σ_4 permet de déduire une première clause assertive (qu'on appellera également Δ_1) car tous ses littéraux sont affectés avant le niveau actuel excepté un (x_{10}) qui est affecté au niveau courant 5 ; $\langle \sigma_1, \sigma_2, \sigma_3, \sigma_4 \rangle$ est une preuve élémentaire par résolution basée sur les conflits, et $\neg x_{10}$ est le premier UIP.

Si on continue un tel processus, on obtient en plus les deux clauses assertives $\Delta_2 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^5_2)$, correspondant au second UIP $\neg x^5_2$; et $\Delta_3 = (x^1_{17} \vee \neg x^3_{19} \vee \neg x^2_8 \vee \neg x^3_4 \vee x^2_{13} \vee x^1_6 \vee \neg x^5_{11})$, correspondant au 3ième UIP ($\neg x^5_{11}$) mais également dernier UIP puisque x_{11} est la dernière variable de décision (voir coupes 2 and 3 dans la figure 2.1).

Propriété 3 (Clause assertive et retour-arrière) *Soit $c = (\alpha \vee x)$ une clause assertive déduite à partir de l'analyse du conflit et $i = max\{l(\neg y) \mid y \in \alpha\}$. On peut sans danger faire un retour-arrière au niveau i et considérer la nouvelle interprétation partielle $\rho^i \cup \{x\}$*

Propriété 4 *Soit $c = (\alpha \vee x)$ une clause assertive déduite à partir de l'analyse du conflit et $i = max\{l(\neg y) \mid y \in \alpha\}$. alors x peut être déduite par réfutation au niveau i, i.e, $(\mathcal{F} \wedge \overline{x})|_{\rho^i} \models_* \bot$.*

Preuve : L'affectation de tous les littéraux $y \in \alpha$ à faux conduit exactement au même conflit par propagation unitaire.

La propriété précédente montre que le littéral assertif peut être déduit par réfutation au niveau i. Cela signifie que si on décide de tester par propagation unitaire à chaque niveau un ensemble de littéraux (lookahead local), alors on peut détecter ce conflit bien avant d'atteindre un niveau plus bas dans l'arbre de recherche.

La vraie difficulté, cependant, réside dans la manière de sélectionner efficacement les littéraux à pré–traiter par propagation unitaire. Une tentative de réponse à cette question est donnée dans les approches proposées par Dubois et al [DUBOIS *et al.* 1996] et par Li et Anbulagan [LI & ANBULAGAN 1997].

Cette propriété simple montre que les solveurs SAT modernes explorent un arbre de recherche binaire, et peuvent être considérés comme une variante de la procédure DPLL. Ce point de vue n'est pas partagé par tous les membres de la communauté SAT.

2.4 Preuve d'optimalité du schéma "First UIP"

Exemple 5 *On considère une formule $\mathcal{F}$ contenant, entre autres, les clauses suivantes ($\mathcal{F} \supseteq \{c_1, \ldots, c_9\}$) :*

$$
\begin{array}{llll}
(c_1) & x_6 \vee \neg x_{11} \vee \neg x_{12} & (c_2) & \neg x_{11} \vee x_{13} \vee x_{16} \\
(c_3) & x_{12} \vee \neg x_{16} \vee \neg x_2 & (c_4) & \neg x_4 \vee x_2 \vee \neg x_{10} \\
(c_5) & \neg x_8 \vee x_{10} \vee x_1 & (c_6) & x_{10} \vee x_3 \\
(c_7) & x_{10} \vee \neg x_5 & (c_9) & \neg x_3 \vee \neg x_{19} \vee \neg x_{18} \\
(c_8) & x_{17} \vee \neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18} &&
\end{array}
$$

Soit ρ l'interprétation partielle suivante $\rho = \{\langle \ldots \neg x_6^1 \ldots \neg x_{17}^1 \rangle \langle (x_8^2) \ldots \neg x_{13}^2 \ldots \rangle \langle (x_4^3) \ldots x_{19}^3 \ldots \rangle \ldots \langle (x_{11}^5) \ldots \rangle\}$. Le niveau de décision courant est 5. Le graphe d'implication $\mathcal{G}_{\mathcal{F}}^{\rho}$ associé à $\mathcal{F}$ et ρ est schématisé dans la figure 2.2 (notez que seule une partie du graphe est affichée).

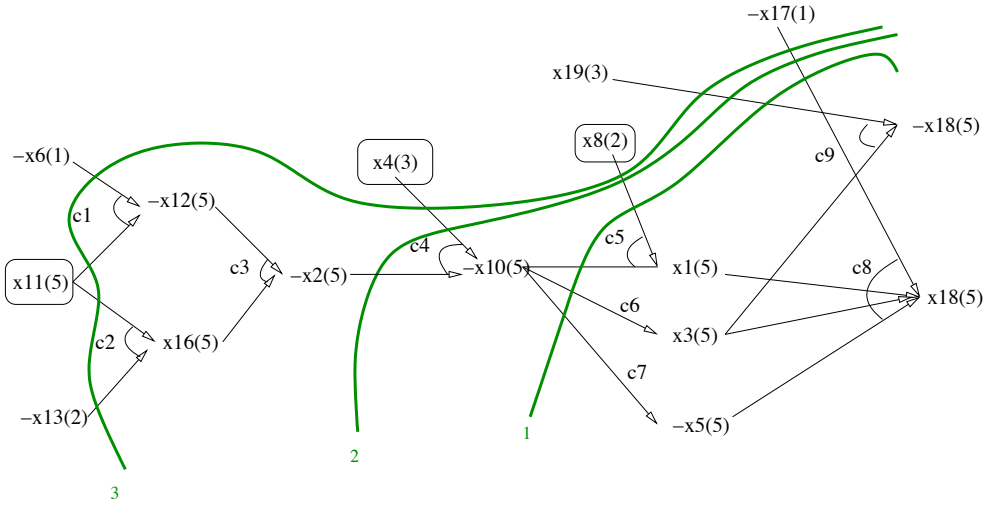


FIG. 2.2 – Graphe d'implication $\mathcal{G}_{\mathcal{F}}^{\rho} = (\mathcal{N}, \mathcal{E})$

Pour motiver notre extension, nous prouvons dans cette section une propriété fondamentale concernant l'optimalité de la clause assertive générée grâce au premier UIP du schéma d'apprentissage : il est optimal en terme de niveau de backjumping[1]. Cette simple propriété montre pourquoi le premier UIP habituellement considéré dans les schémas d'apprentissage classique est plus puissant que les autres UIPs.

Soit c une clause, on note $levels(c)$ l'ensemble des différents niveaux présents dans la clause c, i.e, $levels(c) = \{l(x) \mid x \in c\}$.

[1] Après discussions privées [SILVA & SAKALLAH 1996], il apparaît que ce résultat fait partie du folklore de la communauté, mais, d'après nos connaissances, il semble que ce soit la première fois que cela est formellement établi

Propriété 5 *Dans une preuve par résolution basée sur les conflits $\langle c_1, \ldots, c_k \rangle$ on a pour tout $i < k$, $levels(c_i) \subseteq levels(c_{i+1})$.*

Preuve : Chaque σ_{i+1} est obtenu à partir de σ_i en sélectionnant un littéral $x \in \sigma_i$ qui n'est pas un littéral de décision et en le remplaçant par l'ensemble de ces explications $exp(x)$. Ces explications contiennent toujours d'autres littéraux y tel que $l(y) = l(x)$.

Propriété 6 *Soit $c = (\alpha, x)$ une clause assertive obtenue par preuve par résolution sur le conflit et dans laquelle x est le premier UIP. Le niveau du backjumping de c est optimal : toute autre assertive clause c' obtenue par résolution sur les conflits est telle que $jump(c') \geq jump(c)$.*

Preuve : Il peut être démontré que toutes les clauses assertives contenant le premier UIP ont le même niveau du backjump (en général, il peut y avoir plusieurs clauses de telle sorte, mais leurs ensembles de niveaux peuvent être incomparables). Soit $\langle c_1, \ldots, c' \rangle$ une résolution basée sur les conflits de c'. Soit i l'index de la première clause assertive dans cette preuve. Comme c_i est une clause assertive, on a $jump(c) = jump(c_i)$ et en utilisant la propriété 5, on a $jump(c_i) \leq jump(c')$.

Notons que le premier UIP est le seul qui garantit ces deux points d'optimalité. On peut construire aisément des exemples dans lesquels n'importe quelle clause assertive associée à un UIP autre que le premier a un niveau de backjump strictement plus faible que celui du premier UIP.

2.5 eCDCL : un cadre étendu pour l'analyse de conflits

2.5.1 Graphe d'implications étendu

On montre dans cette section comment les arcs provenant des clauses satisfaites peuvent être ajoutés au graphe d'implication et peuvent être exploités avantageusement [2]. En particulier, ces arcs permettent parfois d'améliorer le niveau du backjump. Comme montré dans la section précédente, c'est quelque chose d'impossible si l'on considère le graphe d'implication classique.

Illustration Dans les solveurs SAT modernes, seule la première clause $exp(x)$ rencontrée et impliquant un littéral x est prise en compte dans l'analyse. Or ce même littéral peut être impliqué par d'autres clauses et peut donc posséder plusieurs explications. Cependant ces dernières sont ignorées dans les solveurs. Ceci est dû principalement à une question de performances. C'est justement ces clauses qui représentent des arcs particuliers dans le graphe d'implication (qui ne sont pas représentées jusqu'à maintenant) qui nous intéressent et qui constituent la base de notre travail.

Pour expliquer notre idée considérons à nouveau la formule $\mathcal{F}$ de l'exemple 5 On considère exactement la même interprétation partielle et on définit la nouvelle formule $\mathcal{F}'$ comme suit : $\mathcal{F}' \supseteq \{c_1, \ldots, c_9\} \cup \{c_{10}, c_{11}, c_{12}\}$ avec $c_{10} = (\neg x_{19} \vee x_8), c_{11} = (x_{19} \vee x_{10})$ et $c_{12} = (\neg x_{17} \vee x_{10})$.

Les trois clauses ajoutées sont satisfaites par l'interprétation ρ. c_{10} est satisfaite par x_8 affecté au niveau 2, c_{11} est satisfaite par le littéral x_{19} affecté au niveau 3 quand à c_{12} elle est satisfaite par l'affectation de $\neg x_{17}$ au niveau 1. Ceci est décrit dans le graphe d'implication étendu de la figure 2.3 grâce aux arcs en pointillé. Illustrons maintenant l'utilité de notre extension proposée. Considérons une nouvelle fois la clause assertive Δ_1 qui correspond au premier UIP classique.

On peut générer la clause assertive forte suivante :

– $c_{13} = \eta[x_8, \Delta_1, c_{10}] = (x_{17}^1 \vee \neg x_{19}^3 \vee x_{10}^5)$

[2] Un notion de graphe d'implication étendu dans lequel plusieurs clauses "explicatives" sont maintenues pour chaque littéral a déjà été introduite dans la littérature [BEAME *et al.* 2004]. Néanmoins, Le but de ce travail était théorique alors que le nôtre est de montrer que les arcs arrière ont un intérêt pratique

– $c_{14} = \eta[x_{19}, c_{13}, c_{11}] = (x^1_{17} \vee x^5_{10})$
– $\Delta^s_1 = \eta[x_{17}, c_{14}, c_{12}] = x^5_{10}$

Dans ce cas on saute au niveau 0 et on affecte x_{10} à $vrai$. En effet $\mathcal{F}' \models x_{10}$.

Comme on peut le voir Δ^s_1 subsume Δ_1. Si on continue le processus, on obtient d'autres clauses assertives fortes $\Delta^s_2 = (\neg x^3_4 \vee x^5_2)$ et $\Delta^s_3 = (\neg x^3_4 \vee x^2_{13} \vee x^1_6 \vee \neg x^5_{11})$ qui subsument respectivement Δ_2 et Δ_3. Cette première illustration permet d'obtenir une nouvelle voie pour minimiser la taille de la clause assertive.

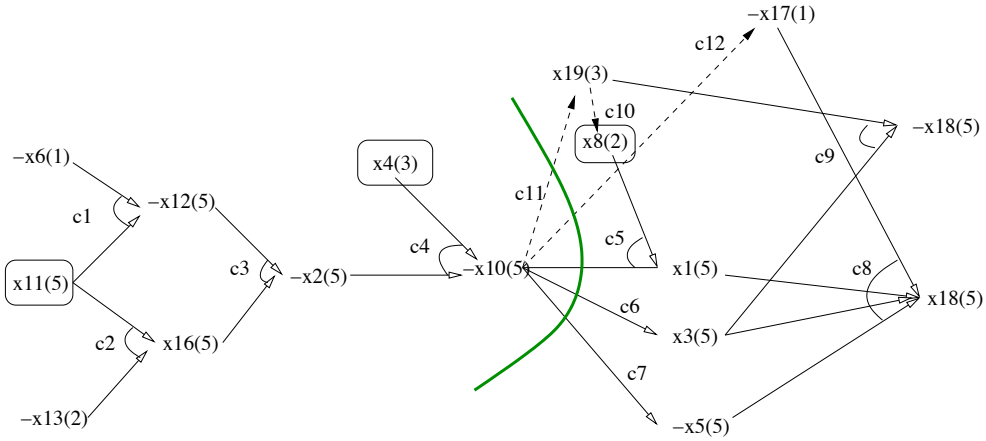


FIG. 2.3 – Graphe d'implication étendu : $\mathcal{G}s^{\rho}_{\mathcal{F}'} = (\mathcal{N}', \mathcal{E}')$

Si nous jetons un coup d'œil aux clauses utilisées dans le graphe d'implication classique $\mathcal{G}^{\rho}_{\mathcal{F}}$ (figure 2.2), elles ont toutes les propriétés suivantes : (1) $\forall x \in \mathcal{N}$ la clause $c = (\overline{exp(x)} \vee x)$ est satisfaite par seulement un littéral i.e, $\rho(x) = vrai$ et $\forall y \in exp(x)$, On a $\rho(y) = vrai$ et (2) $\forall y \in exp(x)$, $l(\neg y) \leq l(x)$. Maintenant dans le graphe d'implication étendu (figure 2.3) les clauses ajoutées satisfont la propriété (1) et, en plus, la propriété (2') $\exists y \in exp(x)$ avec $l(\neg y) > l(x)$.

Expliquons brièvement comment ces arcs supplémentaires sont obtenus. Habituellement, la propagation unitaire ne garde pas de trace des implications apparaissant dans la sous-formule satisfaite par l'interprétation. Dans cette extension, on prend en compte la sous-formule satisfaite pour déduire de nouvelles implications. Revenons à notre exemple, quand on déduit x_{19} au niveau 3, on découvre l'implication de x_8 (qui est le littéral de décision du niveau 2). De manière similaire, pour $\neg x_{10}$ déduit au niveau 5, on "redécouvre" les déductions x_{19} (faites au niveau 3) et $\neg x_{17}$ (faites au niveau 1).

Notre approche tient compte de ces déductions "redécouvertes". Notez quelque chose d'inhabituel : les points de choix peuvent également avoir des arcs entrants (comme x_8 dans l'exemple), ce qui est impossible avec un graphe d'implication classique.

Avant d'introduire formellement la définition du graphe d'implication étendu, on va introduire le concept d'implication arrière (arc arrière).

On maintient en plus de la clause classique $\overrightarrow{cla}(x)$ une nouvelle clause $\overleftarrow{cla}(x)$ de la forme $(x \vee y_1 \vee \cdots \vee y_n)$. Cette clause est sélectionnée de la façon suivante $\rho(y_i) = faux$ pour tout $i \in 1 \ldots n$; $\rho(x) = vrai$; et $\exists i.\ l(y_i) > l(x)$. Cette clause peut être indéfinie dans certains cas (ce qu'on note $\overleftarrow{cla}(x) = \bot$). Plusieurs clauses de cette forme peuvent être trouvées pour chaque littéral. Dans ce cas, une seule est sélectionnée arbitrairement : on peut par exemple choisir de considérer la première trouvée. (on peut toutefois définir des variantes si on veut tenir compte de toutes ces clauses, dans ce cas $\overleftarrow{cla}(x)$ est l'ensemble des clauses ; mais le but ici n'est pas de développer cette variante).

On note par $\overleftarrow{exp}(x)$ l'ensemble $\{\overline{y} \mid y \in \overleftarrow{cla}(x) \setminus \{x\}\}$, et, par souci de clarté, par $\overrightarrow{exp}(x)$ l'ensemble

précédemment noté exp. Un graphe d'implication étendu est défini comme suit (notons que ce graphe n'est pas acyclique dans le cas général) :

Définition 32 (Graphe d'implication étendu) *Soit $\mathcal{F}$ une formule CNF et ρ une interprétation partielle ordonnée. On définit le graphe d'implication étendu associé à $\mathcal{F}$ et ρ comme $\mathcal{G}s^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E} \cup \mathcal{E}')$ tel que,*

- $\mathcal{N} = \rho$
- $\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overrightarrow{exp}(y)\}$, $\mathcal{E}' = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overleftarrow{exp}(y)\}$

2.5.2 Génération de clauses assertives étendues

Dans cette section, nous décrivons une première extension possible de l'approche CDCL utilisant le graphe d'implication étendu. Notre approche fait une utilisation originale des arcs arrière afin d'effectuer des sauts arrière intelligents plus importants, c'est-à-dire d'améliorer le niveau du backjumping généré par la clause assertive. Illustrons tout d'abord l'idée principale derrière cette extension. Notre approche est réalisée en trois étapes. Dans la première étape (1) : une clause assertive, dite $\sigma_1 = (\neg x^1 \vee \neg y^3 \vee \neg z^7 \vee \neg a^9)$ est apprise à partir du schéma d'apprentissage classique, le niveau de décision actuel est donc égal à 9. Comme $\rho(\sigma_1) = faux$, habituellement on effectue un retour-arrière au niveau $jump(\sigma_1) = 7$. Dans la seconde étape (2), notre approche vise à éliminer le littéral $\neg z^7$ à partir de σ_1 en utilisant un arc arrière du graphe étendu. Expliquons à présent cette deuxième et nouvelle transformation. Soit $c = (z^7 \vee \neg u^2 \vee \neg v^9)$ tel que $\rho(z) = vrai$, $\rho(u) = vrai$ et $\rho(v) = vrai$. La clause c est un arc arrière i.e, le littéral z affecté au niveau 7 est impliqué par les deux littéraux u et v affectés respectivement au niveau 2 et 9. A partir de c et de σ_1, une nouvelle clause $\sigma_2 = \eta[z, c, \sigma_1] = (\neg x^1 \vee \neg u^2 \vee \neg y^3 \vee \neg v^9 \vee \neg a^9)$ est générée par résolution. On peut remarquer que la nouvelle clause σ_2 contient deux littéraux du niveau de décision actuel (9). En troisième étape (3), en utilisant le graphe d'implication classique, on peut faire une preuve par résolution basée sur les conflits à partir de σ_2 pour trouver une nouvelle clause assertive σ_3 avec un seul littéral du niveau de décision actuel.

Notons que cette nouvelle clause assertive σ_3 peut dégrader le niveau du backjumping. Pour éviter cet inconvénient majeur, l'arc arrière c est choisi sous conditions : i) les littéraux de c qui sont du niveau de décision actuel (v^9) sont déjà visités durant la première étape. et ii) tous les autres littéraux de c sont affectés avant le niveau 7 (niveau de z). Dans ce cas, on garantit que la nouvelle clause assertive satisfait la propriété suivante : $jump(\sigma_3) \leq jump(\sigma_1)$. Plus intéressant encore, le littéral assertif de σ_3 est $\neg a$.

On peut réitérer le processus précédant sur la nouvelle clause assertive σ_3 pour éliminer les littéraux de σ_3 affectés au niveau $jump(\sigma_3)$.

Afin d'introduire formellement notre approche, nous introduisons dans ce qui suit le concept d'arc arrière joint pour étendre la résolution basée sur les conflits classique.

Définition 33 (Arc arrière joint) *Soit $\pi = \langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ une preuve par résolution basée sur les conflits. Une clause $c \in \mathcal{F}$ est appelée arc arrière joint de π ssi les conditions suivantes sont satisfaites :*

1. *c est l'arc arrière associé à un des littéraux de σ_k dont le niveau est le niveau du backjumping : $\exists x \in \sigma_k$ tq. $c = \overleftarrow{cla}(\overline{x})$ et $l(x) = jump(\sigma_k)$;*
2. *$\forall y \in c$, soit $l(y) = m$ soit $l(y) < jump(\sigma_k)$.*

Quand il existe un arc arrière joint c pour une preuve de résolution basée sur les conflits π, cela signifie qu'on peut l'utiliser pour *étendre* cette preuve : on obtient la clause $\sigma_{k+1} = \eta[x, \sigma_k, c]$, à partir de laquelle on commence une nouvelle preuve $\pi' = \langle \sigma_{k+1} \ldots \sigma_\iota \rangle$ (Intuitivement c "joint" π et π'). Cet ensemble de séquences est appelé preuve de résolution jointe basée sur les conflits :

Définition 34 (preuve par résolution jointe) *Une preuve par résolution jointe basée sur les conflits est une séquence de clauses $\langle \sigma_1 \dots \sigma_k, \sigma_{k+1}, \dots \sigma_\iota \rangle$ telle que*

- *$\langle \sigma_1 \dots \sigma_k \rangle$ est une preuve de résolution basée sur les conflits ;*
- *$\sigma_{k+1} = \eta[x, \sigma_k, c]$, tel que c est un arc arrière joint ;*
- *Les clauses $\sigma_{k+2}, \dots \sigma_\iota$ sont utilisées comme preuve de résolution basée sur les conflits, i.e, elles respectent les conditions (2) et (3) de la définition 27.*

Clairement, à partir de la définition 34, on peut dériver une nouvelle clause assertive. L'étape de jointure (appliquer la résolution entre σ_k et c) produit une nouvelle clause telle que le littéral qui a le plus haut niveau est éliminé. A partir de la nouvelle clause générée σ_{k+1}, on peut dériver par résolution (en utilisant les arcs du graphe d'implication classique) une autre clause conflit σ_ι. Pour être sur que la nouvelle clause σ_ι admette un niveau de backjump inférieur ou égal à $jump(c_k)$, une nouvelle condition sur l'arc arrière joint est introduite :

Définition 35 *Soit $\pi = \langle \sigma_1, \sigma_2, \dots \sigma_k \rangle$ une preuve par résolution basée sur les conflits. On définit $resLit(\pi)$ comme étant l'ensemble des littéraux du dernier niveau de décision m utilisés dans les résolutions de π*

Propriété 7 *Soit $\pi^e = \langle \sigma_1 \dots \sigma_k, \sigma_{k+1}, \dots \sigma_\iota \rangle$ une preuve par résolution jointe basée sur les conflits tel que k est la position de l'arc arrière joint c. Si $\forall x \in c$ tq. $l(x) = m$, alors $x \in resLit(\pi_1) \cup \{\mathcal{A}(\sigma_k)\}$ alors $jump(\sigma_\iota) \leq jump(\sigma_k)$.*

Preuve : esquissons la preuve. Comme π_1 est une preuve par résolution basée sur les conflits, $\sigma_k = (\alpha \vee y \vee x)$ avec $\mathcal{A}(\sigma_k) = x$ et $l(\neg y) = jump(\sigma_k)$, est une clause assertive. Sans perte de généralité, supposons que $\forall z \in \alpha, l(\neg z) < l(\neg y)$. Comme π^e est une preuve par résolution basée sur les conflits, $\exists c = (\beta \vee \neg y)$ un arc inverse joint et $\sigma_{k+1} = \eta[y, \sigma_k, c] = (\alpha \vee \beta \vee x)$. Maintenant, à partir de σ_{k+1}, on applique la résolution uniquement sur l'ensemble des littéraux $\beta_1 = \beta \cap resLit(\pi_1)$. Comme ces littéraux sont précédemment utilisés dans la preuve par résolution π_1, alors à partir de σ_{k+1} on peut dériver en suivant les mêmes arcs (par résolution) la clause assertive σ_ι de la forme $(\alpha \vee \beta \backslash \beta_1 \vee x)$ ou $(\alpha \vee \beta \backslash \beta_1 \vee y \vee x)$. Dans le premier cas, $jump(\sigma_\iota) < jump(\sigma_k)$. Par hypothèse et définition de c (voir condition 2), on déduit que les littéraux de α et $\beta \backslash \beta_1$ sont affectés à un niveau inférieur a $l(\neg y)$. Dans le deuxième cas, le littéral y qu'on veut éliminer de σ_k apparaît dans σ_ι. Par conséquent, les deux clauses assertives admettent le même niveau de backjumpings $l(y)$. Supposons que la clause assertive σ_k contient plusieurs littéraux avec le même niveau que y. Dans ce cas, en éliminant un seul littéral, on obtient le même niveau de backjumping.

A partir de la propriété 7, nous avons montré comment améliorer le niveau du backjumping en éliminant le littéral du niveau du backtrack. Naturellement, quand la clause assertive contient plusieurs littéraux du niveau du backtrack, on peut réitérer le processus sur la nouvelle clause assertive pour éliminer d'autres littéraux. Néanmoins, ceci peut poser certaines difficultés que nous détaillons dans la section 2.6.1.

Plus intéressant encore, quand tous les littéraux du niveau le plus grand sont éliminés, on peut itérer le processus sur les littéraux du nouveau plus grand niveau et ainsi faire plusieurs backjumping successifs à partir d'un seul conflit. Néanmoins, en pratique, un tel processus itératif peut nécessiter beaucoup de ressources.

2.6 Mise en œuvre en pratique

2.6.1 Améliorer la qualité du saut arrière

Notre but est donc de faire un saut arrière plus important dans l'arbre de recherche que celui déduit par la clause assertive. A cette fin, nous devons supprimer grâce à des arcs arrière tous les littéraux présents dans la clause assertive et appartenant au même niveau de backjump. Soit $\mathcal{S}_b = \{x_i \in cla | level(x_i) = b\}$ cet ensemble. Deux problèmes se posent alors :

- Tout d'abord, la recherche d'arcs arrière pour tous les littéraux de $\mathcal{S}_b$ va induire un coût non négligeable dans notre algorithme. Ce surcoût peut vite devenir prohibitif vu le nombre de clauses présentes dans les instances industrielles.
- Ensuite, les arcs arrière qui vont être ajoutés à chaque nœud x_i de $\mathcal{S}_b$ du graphe d'implication peuvent le rendre cyclique. Ceci est un problème majeur qui ne peut être évité qu'en recherchant des arcs arrière particuliers.

Pour pallier ces problèmes, nous proposons, dans une première intégration, une restriction sur les arcs arrière. Nous détaillons ceci dans la section suivante.

Solveurs SAT modernes : première intégration

Avant de décrire les restrictions opérées pour intégrer les arcs arrière dans le schéma d'apprentissage, nous souhaitons rappeler que cette intégration peut être effectuée sur n'importe quel solveur de type CDCL. Nous souhaitons rappeler également les composantes essentielles constituant les solveurs SAT modernes : (1) l'analyse du conflit, (2) l'heuristique VISDS (prenant en compte l'activité des littéraux), (3) l'apprentissage des clauses assertives, (4) la politique de redémarrage. L'intégration proposée est la suivante :

1. *Analyse du conflit* : A chaque conflit, l'apprentissage classique est appliqué et une clause assertive $\sigma_1 = (\alpha \vee y^i \vee x^m)$, avec x le littéral assertif et $i = jump(\sigma_1)$, est générée grâce à une preuve par résolution basée sur les conflits π_1 (en utilisant le premier UIP). Avant d'effectuer le retour-arrière au niveau i, on essaie d'utiliser la preuve de résolution jointe basée sur les conflits afin de générer une nouvelle clause assertive σ_2. Dans notre implémentation, cette extension est seulement appliquée si σ_1 contient un seul littéral du niveau i. En général, on peut réitérer le processus pour tous les littéraux du niveau i. Une autre restriction est que l'arc inverse doit contenir des littéraux du niveau m ou de niveau j tel que $j < i$ i.e, $x \in resLit(\sigma_1) \cup \{\mathcal{A}(\sigma_1)\}$. Sans cette restriction, on peut générer d'autres clauses assertives qui peuvent être plus ou moins intéressantes en terme du niveau de backjumping.
2. *Activité des littéraux* : les activités des littéraux introduits par l'arc arrière sont aussi pondérées.
3. *Apprentissage* les deux clauses assertives sont ajoutées à la base des nogoods.

2.6.2 Réduire la taille des clauses assertives

Nous avons vu dans les sections 2.6.1 pourquoi la suppression des tous les littéraux du dernier point de choix était compliquée et comment nous proposions d'y remédier. Nous envisageons dès à présent une autre façon de pallier ce problème. Le but est de se concentrer sur les arcs inverses binaires (de taille 2). D'une part, parce que la détection d'arc inverse binaire est relativement facile, de l'autre, parce que si un arc inverse est exploitable, nous sommes sûrs de réduire la taille de la clause assertive.

Solveurs SAT modernes : seconde intégration

Détection des arcs inverses binaires Expliquons tout d'abord pourquoi, dans les solveurs modernes, la détection des arcs arrière binaires est une opération non coûteuse. Soit $(x_1 \vee x_2)$ une clause binaire. Les deux littéraux de cette clause seront donc "*watchés* par les structures de données paresseuses. S'il est nécessaire de visiter cette clause durant la propagation unitaire, alors l'un des littéraux est $faux$. Si l'autre littéral est $vrai$, c'est qu'il a été déjà propagé et donc cette clause est tout simplement un arc inverse. La détection ne nécessite donc pas de surcoût et peut être directement réalisée durant la phase de propagation.

Exploitation des arcs inverses binaires Comme expliqué précédemment, les arcs inverses exploitables sont des arcs qui mettent en relation un littéral de la clause assertive et un littéral des nœuds visités durant le processus de l'analyse de conflit. Au lieu de sauvegarder pour un littéral l tous les arcs inverses l'impliquant, on va associer à chaque littéral un deuxième littéral $a(l) = l_1$. A chaque fois qu'un littéral est impliqué par un arc inverse l_2, l_1 est écrasé et remplacé par le nouveau arc $a(l) = l_2$. Ceci permet de garder le littéral le plus bas dans la chaîne de propagation. Une fois un conflit détecté et la clause assertive construite, un parcours simple de cette dernière permet de vérifier si les littéraux en question sont susceptibles d'être exploités. Reste qu'on ne peut réduire qu'une partie de la clause constituée de littéraux qui, ensemble, ne créent pas de cycles dans le graphe.

Dans les solveurs modernes, a chaque fois qu'un littéral est impliqué, il est ajouté dans la pile. supposons que l'on ajoute à cette pile un littéral qui est impliqué par un arc inverse. Cette pile peut donc avoir plusieurs occurrences du même littéral. Soit l un littéral apparaissant dans un arc inverse, ce littéral apparaît dans la raison d'un littéral x du graphe d'implication. Si lors de la propagation on a détecté un arc inverse de la forme $(l \vee \neg y)$ ou y est un littéral du graphe d'implication, alors l a été impliqué une première fois en haut de l'arbre et une deuxième fois au niveau du conflit. Le littéral l possède donc deux niveaux, le niveau réel (celui du graphe d'implication classique) et un niveau virtuel correspondant à son arc inverse (noté m). Dans ce cas, si $m < l(x)$ alors le littéral l peut être omis de la clause assertive. Cette restriction, qui n'a pas encore été implémentée, ne permet pas de capturer tous les arcs inverses. Néanmoins, le surcoût induit pour réduire la taille de la clause assertive est quasi inexistant.

2.7 Expérimentations

Les résultats expérimentaux reportés dans cette section sont obtenus sur un Xeon 3.2 GHz (2Go RAM) et ont été réalisés sur un large éventail d'instances (286) issus de la Sat-Race'06 [3] et de la compétition SAT'07 [4] (uniquement les instances industrielles). Toutes les instances sont simplifiées à l'aide du pré-processeur SatELite [EÉN & BIERE 2005]. Le temps CPU est limité à 1800 secondes et les résultats sont reportés en secondes. Nous avons implémenté l'approche proposée dans la section 2.6.1 aux solveurs Minisat [EÉN & SÖRENSSON 2003] et Rsat [PIPATSRISAWAT & DARWICHE 2007b] et proposons une comparaison entre les solveurs originaux et leur version étendue (appelés Minisat^E and Rsat^E).

Les deux graphiques (échelle logarithmique) de la figure 2.4 montrent les comparaisons entre Minisat (figure du haut) et Rsat (figure du bas) par rapport à leur version étendue. Chaque point correspond à une instance. L'axe des abscisses (resp. des ordonnées) correspond au temps CPU t_x (resp. t_y) obtenu sur le solveur étendu (resp. original). Chaque point au-dessus (resp. au-dessous) de la diagonale indique que le solveur étendu est plus rapide (resp. plus lent) que la version originale sur l'instance donnée. La figure 2.4 montre clairement que l'utilisation des arcs arrière améliore les performances de Minisat.

[3] http ://fmv.jku.at/sat-race-2006/
[4] http ://www.satcompetition.org

Cela semble beaucoup moins le cas pour Rsat, mais il est important de noter que Rsat^E est capable de résoudre 5 instances de plus que Rsat original (alors que Minisat^E résout une instance de moins que Minisat original).

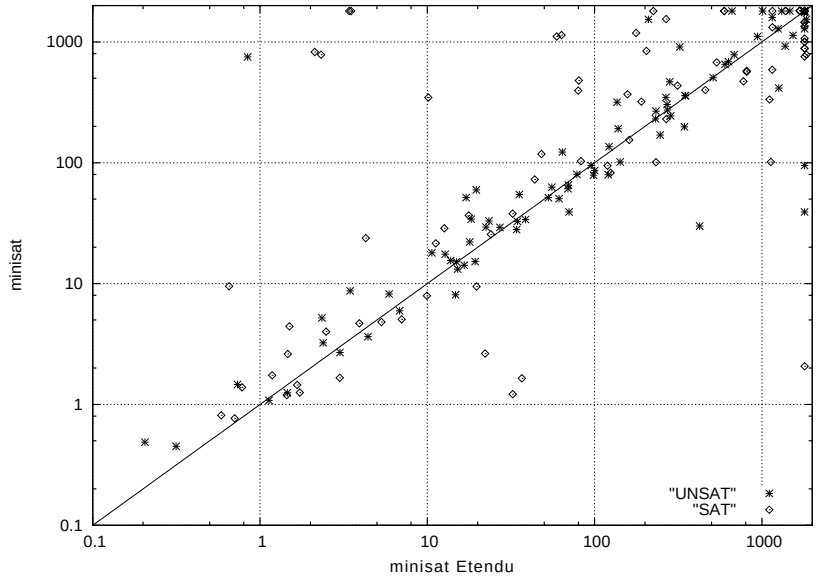
La figure 2.5 montre le temps T (figure du haut) et le temps cumulé CT (figure du bas) nécessaires pour résoudre un nombre donné d'instances ($\#instances$). T représente le nombre d'instances résolues en t secondes, CT représente la même chose mais en cumulant les temps de résolution. Rsat^E et Minisat^E permettent de résoudre efficacement nombreuses instances considérées.

Enfin, la table 2.1 exhibe les résultats obtenus sur une échantillon représentatif des instances. Pour chaque instance, on reporte le temps CPU nécessaire aux 4 différents solveurs.

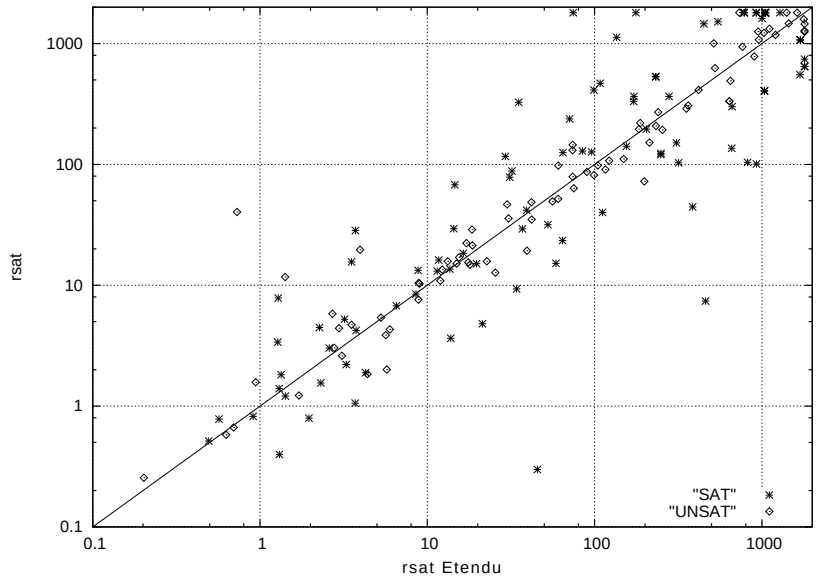
Rappelons que nous appliquons notre méthode uniquement si la clause assertive contient un seul littéral du niveau de backjumping. C'est pour cela que l'utilisation des arcs arrière permet d'améliorer le saut arrière entre 1% et 10 % des conflits. Malgré ce faible pourcentage, notre extension est clairement utile et montre le potentiel des arcs arrière. De plus, sur certaines classes d'instances le temps CPU est amélioré de plusieurs ordres de grandeur. Rsat^E est clairement le meilleur sur les familles `ibm`, `Velev`, `total` et `simon` alors que Minisat^E a de très bons résultats sur les familles `manol`, `vmpc`, `gold`, `APro`.

2.8 Conclusion

Dans ce chapitre, nous avons proposé une généralisation du cadre de l'analyse du conflit. Cette généralisation est obtenue par une extension originale du graphe d'implication classiquement utilisé dans les solveurs CDCL. Cette extension, motivée par les résultats d'optimalité de la clause assertive (Premier UIP) en terme de niveau du backjump, est obtenue en considérant des clauses (appelées arcs inverses) qui proviennent de la partie satisfaite de la formule. Plusieurs schémas d'apprentissage peuvent être définis à partir de cette extension. Une première extension possible de l'apprentissage qui améliore les clauses assertives classiques en terme de retours-arrière intelligents montre le grand potentiel de ce nouveau cadre. Malgré les différentes restrictions, son intégration au sein des solveurs Minisat et Rsat montre des améliorations intéressantes sur les instances industrielles des deux dernières compétitions SAT'07 et SAT-race'06. Nous avons déjà abordé les travaux futurs liés aux arcs inverses dans la section 2.6.2. Notons aussi que nous envisageons de définir d'autres schémas d'apprentissage, par exemple en considérant les arcs inverses comme une alternative à chaque nœud de l'arbre de recherche. En d'autres termes, le but serait d'améliorer les différentes résolvantes générées lors des "preuves par résolution basées sur les conflits" en termes de profondeur de backjumping et/ou de taille de clause. Enfin, nous envisageons d'exploiter les arcs arrière pour minimiser les clauses assertives de façon générale.

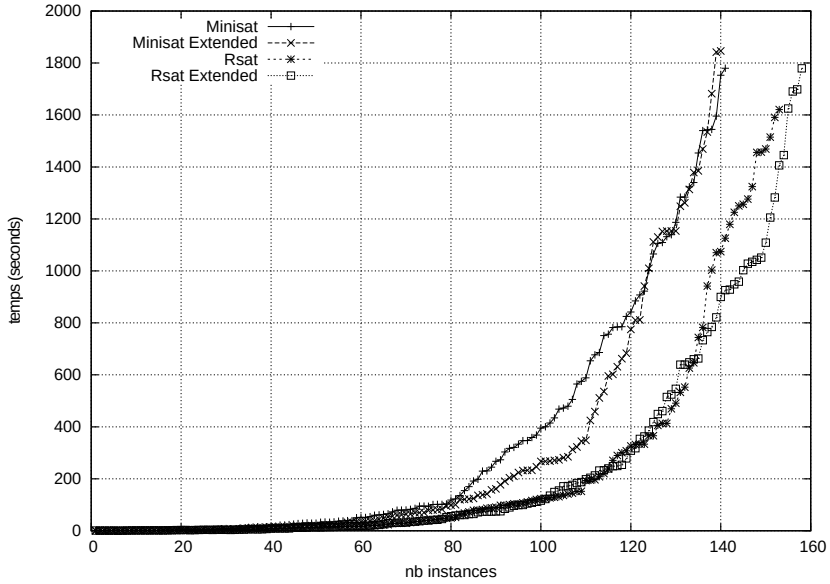


Minisat



Rsat

FIG. 2.4 – Résultats sur les instances de la compétition SAT'07 (industriel) et de la Sat-Race'06



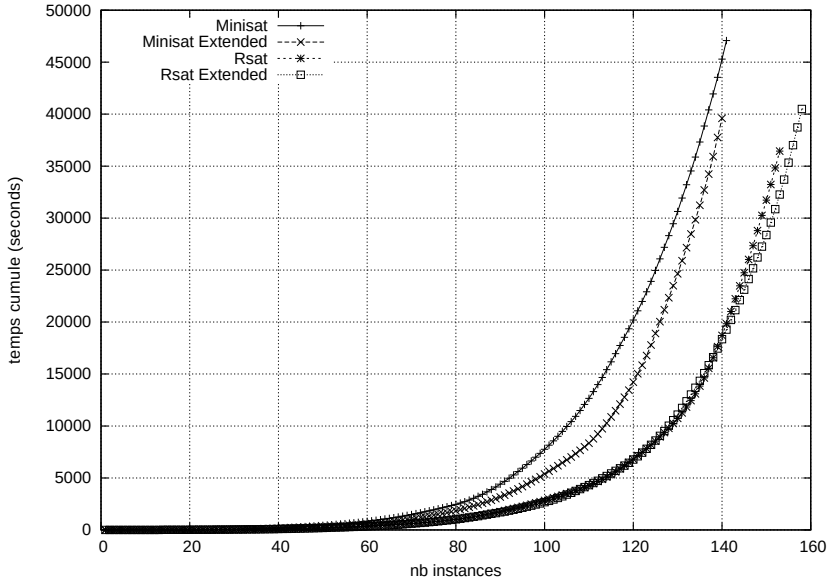


FIG. 2.5 – Temps et temps cumulé pour résoudre un nombre donné d'instances

instance	SAT ?	Rsat^E	Rsat.	Minisat^E	Minisat
AProVE07-02	N	–	–	683.21	782.07
AProVE07-21	N	1406.02	–	511.14	505.41
AProVE07-22	N	232.68	208.17	136.19	316.98
clauses-6	Y	171.21	331.93	810.24	564.78
cube-9-h11-sat	Y	1282.38	–	774.78	472.25
dated-5-15	N	958.61	1074.07	–	1752.71
goldb-heqc-frg2mul	N	187.42	219.99	281.12	468.28
goldb-heqc-i8mul	N	1108.60	1324.17	941.759	1105.80
goldb-heqc-term1mul	N	–	1250.23	–	–
ibm-2002-19r-k100	Y	95.94	126.84	157.31	368.49
ibm-2002-21r-k95	Y	64.68	125.04	268.14	229.77
ibm-2002-26r-k45	N	3.96	19.65	0.84	751.33
IBM_04_30_dat.k8	Y	1042.88	–	1682.10	–
IBM_2004_30_SAT_dat.k100	Y	784.22	–	–	–
IBM_2004_30_SAT_dat.k55	Y	231.99	532.94	–	–
IBM_2004_30_SAT_dat.k60	Y	1698.74	1070.30	595.85	–
IBM_2004_30_dat.k80	Y	925.87	–	–	–
IBM_2004_30_dat.k85	Y	1042.88	–	1682.10	–
IBM_2004_30_dat.k90	Y	1051.16	–	–	–
manol-pipe-c7nidw	N	254.00	193.00	1469.20	–
manol-pipe-f7idw	N	1624.76	–	1010.01	–
manol-pipe-g10bidw	N	–	–	1313.43	–
manol-pipe-g10id	N	73.73	131.10	209.71	1539.87
mizh-md5-47-3	Y	107.98	469.35	1111.21	333.34
mizh-md5-47-4	Y	135.36	1125.72	267.50	1544.60
mizh-sha0-35-4	Y	278.60	365.84	1153.66	1323.37
partial-5-13-s	Y	546.21	1514.82	–	–
schup-l2s-guid-1-k56	N	524.18	626.08	322.68	907.43
simon-s02b-dp11u10	N	515.00	1003.18	284.89	244.64
simon-s02b-r4b1k1.1	Y	1002.87	1620.79	458.83	400.30
total-10-17-s	Y	98.98	413.28	–	–
total-10-19-s	Y	74.46	–	–	–
total-5-17-s	Y	14.59	67.85	1384.98	–
velev-pipe-sat-1.1-b7	Y	70.85	238.16	224.04	–
velev-pipe-uns-1.0-9	N	764.74	941.76	–	–
velev-pipe-uns-1.1-7	N	733.14	–	–	–
vmpc_30	Y	176.49	–	–	–

TAB. 2.1 – Quelques résultats comparatifs

3

Apprentissage à partir des succès

Sommaire

3.1 Introduction 34
3.2 Apprentissage à partir des succès 35
3.2.1 Motivation 35
3.2.2 Présentation Formelle 36
3.3 Intégration de l'approche SDCL dans un solveur SAT moderne 37
3.4 Expérimentations 39
3.5 Conclusion 43

Dans ce chapitre, un nouveau schéma d'apprentissage pour SAT est proposé. L'originalité de cette approche réside dans sa capacité à opérer de l'apprentissage à chaque nœud de l'arbre de recherche et pas uniquement lors d'un conflit. Ceci contraste avec toutes les approches d'apprentissage traditionnelles qui généralement se réfèrent à l'analyse de conflit. Pour rendre cet apprentissage possible, des clauses pertinentes, extraites à partir de la partie satisfaite de la formule, sont utilisées conjointement avec le graphe d'implication classique pour dériver de nouvelles explications plus puissantes pour un littéral donné. Basé sur cette extension, un premier schéma appelé apprentissage à partir des succès ("Success Driven Clause Learning (SDCL)") est proposé. Des résultats expérimentaux montrent que l'intégration de SDCL dans un solveur SAT moderne comme Minisat permet d'obtenir des résultats intéressants particulièrement sur les instances satisfiables.

3.1 Introduction

Dans cette partie, nous nous occupons de l'apprentissage dans les solveurs SAT. Notre but est de montrer qu'il est possible d'apprendre non seulement à partir des échecs (ou conflits) mais aussi à partir des succès. Comme l'apprentissage renvoi à l'analyse de conflits, le cadre proposé ici diffère des approches classiques d'apprentissage au cours de la recherche bien connues dans le cadre SAT et CSP.

Usuellement, à un nœud donné de l'arbre de recherche, lorsqu'un littéral x est affecté par propagation unitaire au niveau i, la clause à l'origine de la propagation, usuellement appelée implication, est intégrée dans le graphe d'implication. Cette clause est de la forme $(\alpha \rightarrow x)$ où α est l'explication de la propagation du littéral unitaire x au niveau i. Il est important de noter que l'explication α inclut au moins un littéral affecté au niveau i.

L'approche que nous proposons ici vise à découvrir de nouvelles explications pour l'implication de x incluant seulement des littéraux des niveaux précédents ($j < i$). En d'autres termes, notre approche est capable de découvrir qu'un littéral affecté au niveau i, peut être déduit dans les niveaux supérieurs $j < i$.

En pratique, ces nouvelles raisons quand elles sont ajoutées dans le graphe d'implication peuvent améliorer le processus d'analyse de conflit lui-même.

Le chapitre est organisé comme suit. Après la présentation de notre approche d'apprentissage à partir des succès dans la section 3.2, une première intégration dans les solveurs SAT modernes est ensuite proposée (section 3.3). Finalement avant de conclure, nous présentons les premiers résultats expérimentaux démontrant les performances de notre approche.

3.2 Apprentissage à partir des succès

Dans cette section, nous montrons comment de nouvelles explications pour l'implication des littéraux (littéraux propagés unitairement) peuvent être générées si une interprétation partielle ne mène pas à un conflit. Ces nouvelles capacités offertes par cette approche sont motivées dans l'exemple suivant.

3.2.1 Motivation

Exemple 6 *Soit la formule $\mathcal{F}$ contenant les clauses $\{c_1, \ldots, c_9\}$ suivantes :*

(c_1)	$x_6 \vee \neg x_{11} \vee \neg x_{12}$	(c_2)	$\neg x_{11} \vee x_{13} \vee x_{16}$	(c_3)	$x_{12} \vee \neg x_{16} \vee \neg x_2$
(c_4)	$\neg x_4 \vee x_2 \vee \neg x_{10}$	(c_5)	$\neg x_8 \vee x_{10} \vee x_1$	(c_6)	$x_{10} \vee x_3$
(c_7)	$x_{10} \vee \neg x_5$	(c_8)	$x_{17} \vee \neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18}$	(c_{10})	$x_6 \vee \neg x_{10} \vee x_{18}$

En considérant l'interprétation courante
$\rho = \{\langle \ldots \neg x_6^1 \ldots \neg x_{17}^1 \rangle \langle (x_8^2) \ldots \neg x_{13}^2 \ldots \rangle \langle (x_4^3) \ldots x_{19}^3 \ldots \rangle \ldots \langle (x_{11}^5) \ldots \rangle\}$. *Le graphe d'implication associé à cette interprétation est donné dans la figure 3.1.*

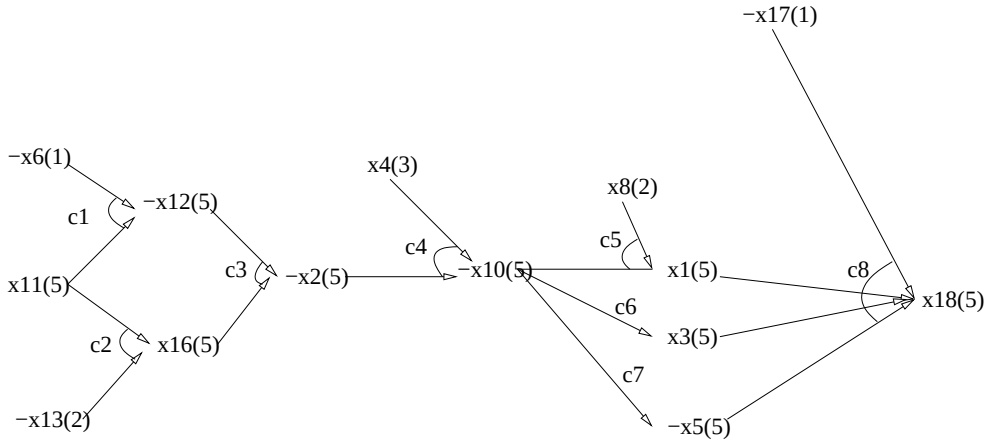


FIG. 3.1 – Graphe d'implication $\mathcal{G}_{\mathcal{F}}^{\rho}$ de l'exemple 6

L'interprétation partielle ρ ne conduit pas à un conflit au niveau de décision actuel (égal à 5). Nous pouvons remarquer que la clause c_{10} n'est pas mémorisée dans le graphe d'implication $\mathcal{G}_{\mathcal{F}}^{\rho}$ car elle contient deux littéraux $\neg x_{10}$ et x_{18} affectés à $vrai$. Notons également que le littéral x_{18} est affecté par propagation unitaire au niveau 5 grâce à la clause c_8. Illustrons maintenant comment nous pouvons déduire une nouvelle implication exprimant le fait que x_{18} aurait du être propagé au niveau 2. En partant de la clause $\overrightarrow{cla}(x_{18})$, les résolvantes suivants sont générées :

- $\sigma'_1 = \eta[x_1, c_8, c_5] = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee \neg x_3^5 \vee x_5^5 \vee x_{18}^5)$
- $\sigma'_2 = \eta[x_3, \sigma'_1, c_6] = (x_{17}^1 \vee \neg x_8^2 \vee x_{10}^5 \vee x_5^5 \vee x_{18}^5)$

– $\sigma'_3 = \Delta'_1 = \eta[x_5, \sigma'_2, c_7] = (x^1_{17} \vee \neg x^2_8 \vee x^5_{10} \vee x^5_{18})$

La clause Δ'_1 contient deux littéraux du dernier niveau de propagation (5) qui sont x_{10} et x_{18} et tous ces littéraux sont affectés à $faux$ sauf x_{18}. Maintenant, si nous appliquons une résolution de plus entre Δ'_1 et $c_{10} = (x_6 \vee \neg x_{10} \vee x_{18}) \in \mathcal{F}$, nous obtenons une nouvelle clause assertive $\Delta''_1 = (x^1_6 \vee x^1_{17} \vee \neg x^2_8 \vee x^5_{18})$. Cette dernière clause exprime que le littéral x_{18} affecté au niveau 5 aurait pu être affecté au niveau 2.

En effet, comme $\mathcal{F}' \models \Delta''_1$ et les deux littéraux x_6 et x_{17} (resp. le littéral $\neg x_8$) sont affectés à faux au niveau 1 (resp. niveau 2), on déduit que le littéral x_{18} affecté au niveau 5 aurait pu être propagé au niveau 2 grâce à la raison Δ''_1.

A partir de l'exemple précédant et contrairement au schéma classique d'apprentissage, nous pouvons déjà faire les remarques suivantes :

– σ'_3 n'est pas une clause conflit i.e, $\rho(\sigma'_3) = vrai$
– le littéral x_{18} n'est pas réfuté i.e, la clause déduite Δ''_1 mentionne que x_{18} devrait être affecté à la même valeur $vrai$ au niveau 2 au lieu du niveau 5

3.2.2 Présentation Formelle

Donnons à présent une présentation formelle du schéma d'apprentissage à partir des succès. Dans toutes les définitions et propriétés suivantes, on considère $\mathcal{F}$ une formule CNF, ρ une interprétation partielle telle que $(\mathcal{F}|_\rho)^* \neq \bot$ et m le niveau de décision courant.

Définition 36 (clause binaire assertive) *Une clause $c = (\alpha \vee x \vee y)$ est dite clause binaire assertive ssi $\rho(\alpha) = faux$, $l(\alpha) < m$ et $l(x) = l(y) = m$.*

Définition 37 (preuve par résolution générant une clause assertive binaire) *Soit x un littéral tel que $l(x) = m$ et $\overrightarrow{cla}(x)$ est définie. Une preuve par résolution générant une clause assertive binaire $\pi_b(x)$ est une séquence de clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfaisant les conditions suivantes :*

1. *$\sigma_1 = \overrightarrow{cla}(x)$*
2. *σ_i, pour $i \in 2..k$, est construite en sélectionnant un littéral $y \in \sigma_{i-1}$. La clause σ_i est définie donc comme $\eta[y, \sigma_{i-1}, \overrightarrow{cla}(\bar{y})]$;*
3. *σ_k est une clause binaire assertive.*

Comme le littéral $x \in \sigma_1$ peut être éliminé par résolution, on a $x \in \sigma_k$. En effet, si cette élimination est possible, cela signifie que $\overrightarrow{cla}(x)$ et $\overrightarrow{cla}(\bar{x})$ sont définis. Ceci contredit l'hypothèse $(\mathcal{F}|_\rho)^* \neq \bot$. De plus, x est l'unique littéral de σ_k affecté à $vrai$

Dans la section 2.4, nous avons montré que le littéral assertif x généré en utilisant l'analyse de conflit peut être déduit par propagation unitaire au niveau i, où i est le niveau du saut arrière (ou backjumping) associé à la clause assertive $(\alpha \vee x)$ i.e, $\mathcal{F}|_{\rho^i} \models_* x$. De façon similaire, pour une clause assertive binaire $(\alpha \vee y \vee x)$, la clause binaire $(y \vee x)$ (voir la propriété 8) peut être aussi déduite par propagation unitaire au niveau $i = l(\alpha)$.

Propriété 8 *Soit $\pi_b(x) = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ une preuve par résolution générant une clause assertive binaire telle que $\sigma_k = (\alpha \vee y \vee x)$ où $i = l(\alpha)$. Nous avons $\mathcal{F}|_{\rho^i} \models^* (y \vee x)$*

Preuve : Par définition de la preuve par résolution générant une clause assertive binaire, $\pi_b(x)$ est dérivée en traversant le graphe d'implication associé à $\mathcal{F}$ et ρ à partir du nœud x. Il est évident que l'affectation $\neg y$ au niveau i mène à la propagation du littéral x. En effet, toutes les clauses utilisées dans

la preuve par résolution générant une clause assertive binaire $\pi_b(x)$ contiennent seulement des littéraux de $\bar{\alpha}$ et des littéraux propagés au niveau de décision courant m. En conséquence, on a $\mathcal{F}|_{\rho^i} \wedge \neg y \models^* x$. Donc $\mathcal{F}|_{\rho^i} \wedge \neg y \wedge \neg x \models^* \bot$

Illustrons la propriété 8 en utilisant l'exemple 6. A partir de la clause assertive binaire $\sigma'_3 = (x^1_{17} \vee \neg x^2_8 \vee x^5_{10} \vee x^5_{18})$, il est facile de montrer que la clause $(x^5_{10} \vee x^5_{18})$ peut être déduite par propagation unitaire au niveau 2 i.e, $\mathcal{F}|_{\rho^2} \models^* (x_{10} \vee x_{18})$. A partir de la formule $\mathcal{F}$ et ρ, on peut voir que la formule $\mathcal{F}|_{\rho^2}$ contient les clauses suivantes $\{(x_{10} \vee x_1), (x_{10} \vee x_3), (x_{10} \vee \neg x_5), (\neg x_1 \vee \neg x_3 \vee x_5 \vee x_{18})\}$ (voir le graphe d'implication $\mathcal{G}^{\rho}_{\mathcal{F}'}$). Donc, on a $\mathcal{F}|_{\rho^2} \wedge \neg x_{10} \models^* x_{18}$. Par conséquent, $\mathcal{F}'|_{\rho^2} \wedge \neg x_{10} \wedge \neg x_{18} \models^* \bot$.

Pour dériver une nouvelle raison pour l'implication d'un littéral x donné on procède en deux étapes. Dans la première étape, nous générons une preuve par résolution générant une clause assertive binaire $\pi_b(x) = \langle \sigma_1, \sigma_2, \ldots, \sigma_k = (\alpha \vee y \vee x) \rangle$, et dans une seconde étape nous éliminons y de σ_k par résolution. Ce processus de résolution est appelé preuve par résolution générant une clause assertive, et est formellement défini ci-dessous :

Définition 38 (preuve par résolution générant une clause assertive) *Soit $\pi_u(x)$ une séquence de clauses $\langle \sigma_1, \ldots, \sigma$ $\pi_u(x)$ est une preuve par résolution générant une clause assertive si elle satisfait les deux conditions suivantes :*

1. *$\langle \sigma_1, \ldots, \sigma_{k-1} = (\alpha \vee y \vee x) \rangle$ est une preuve par résolution générant une clause assertive binaire*
2. *$\exists c = (\beta \vee \neg y \vee x) \in \mathcal{F}$ une clause assertive binaire $\sigma_k = \eta[y, \sigma_{k-1}, c] = (\gamma \vee x)$ où $\gamma = \alpha \cup \beta$.*

Il suit de la définition 36 que tous les littéraux de α et β sont affectés à $faux$ avant le niveau m. Par conséquent, la résolvante σ_k est une clause assertive unitaire. Cela signifie que lorsque $\pi_u(x)$ existe, on déduit que le littéral x propagé au niveau m aurait dû être propagé à $vrai$ au niveau $k < m$ où $k = l(\gamma)$. La propriété suivante fait état de ce résultat.

Propriété 9 *Soit x un littéral tel que $l(x) = m$ et $\overrightarrow{cla}(x)$ existe. Si $\pi_u(x) = \langle \sigma_1, \ldots, \sigma_{k-1}, \sigma_k = (\gamma \vee x) \rangle$ est une preuve par résolution générant une clause assertive alors $\mathcal{F}|_{\rho^i} \models^* x$ où $i = l(\gamma)$.*

Preuve : Premièrement, comme $\langle \sigma_1, \ldots, \sigma_{k-1} \rangle$ est une preuve par résolution générant une clause assertive binaire, la résolvante σ_{k-1} est de la forme $(\alpha \vee y \vee x)$ où $l(\alpha) \leq i$. En utilisant la propriété 8, on peut déduire que $\mathcal{F}|_{\rho^i} \models^* (y \vee x)$, alors $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* y$. De plus, par définition de $\pi_u(x)$, la résolvante σ_k est obtenue par résolution sur y entre σ_{k-1} et $c \in \mathcal{F}$ de la forme $(\beta \vee \neg y \vee x)$ où $l(\beta) \leq i$. Comme $\forall z \in \beta\ \rho(z) = faux$, donc $(\neg y \vee x) \in \mathcal{F}|_{\rho^i}$ et $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* \neg y$. Par conséquent, $\mathcal{F}|_{\rho^i} \wedge \neg x \models^* \bot$.

Dans la propriété 9, nous avons montré que le littéral x peut être déduit par propagation unitaire au niveau i. Cependant, vérifier si chaque littéral non affecté peut être déduit par propagation unitaire à chaque nœud de l'arbre de recherche est coûteux. Dans la section suivante, on montre comment quelques unes de ces déductions peuvent être obtenues à la volée en utilisant notre approche d'apprentissage à partir des succès.

3.3 Intégration de l'approche SDCL dans un solveur SAT moderne

Nous présentons dans cette section comment l'approche SDCL peut être utilisée pour améliorer en pratique les implications d'un solveur SAT moderne. L'amélioration du solveur SAT est esquissée dans l'algorithme 4. Cet algorithme est basé sur l'algorithme CDCL (algorithme 3 vu dans le chapitre 1). Nous rappelons néanmoins les diverses fonctions qui y sont incorporées :

– *propagate* : effectue la propagation unitaire et retourne $faux$ en cas de conflit et $vrai$ sinon.

Algorithm 4: Procédure SDCL

Input: A CNF formula $\mathcal{F}$;
Output: $vrai$ if $\mathcal{F}$ est satisfiable, $faux$ sinon

```
1  begin
2      currentLevel = 0;
3      while (vrai) do
4          if ( !propagate()) then
5              if (currentLevel==0) then return faux;
6              c=(α ∨ ¬d)=analyze();
7              learn(c);
8              backJumpLevel = l(α);
9              bactrack(backJumpLevel+1);
10             newJumpLevel = learnFromSuccess(backJumpLevel+1);
11             bactrack(newJumpLevel);
12         end
13         else
14             currentLevel++;
15             if ( !decide()) then
16                 return vrai
17             end
18         end
19     end
20 end
```

- *analyze* : retourne la clause apprise c calculée en utilisant le schéma classique d'apprentissage (voir chapitre 1). Le littéral assertif est dénoté d.
- *learn* : ajoute une clause apprise c à la base des clauses apprises
- *learnFromSuccess* : applique le schéma d'apprentissage, à partir des succès, présenté dans l'algorithme 5
- *backtrack* : retour-arrière au niveau donné en paramètre.
- *decide* : sélectionne et affecte le littéral de décision suivant. La fonction retourne $vrai$, si la formule est satisfaite (tous les littéraux sont affectés), et $faux$ sinon.

Comme on l'a vu dans la section précédente, l'apprentissage à partir des succès peut être appliqué à chaque nœud de l'arbre de recherche. En d'autres termes, pour chaque littéral unitaire y propagé à un certain niveau i, on peut appliquer le schéma d'apprentissage à partir des succès pour déduire une implication à un niveau précédent i. Cependant, une application systématique peut réduire les performances des solveurs. Dans l'algorithme 4, on applique l'apprentissage à partir des succès (ligne 10) seulement lorsqu'un conflit est détecté i.e, si l'appel à *propagate()* retourne $faux$. Plus précisément, on utilise le schéma classique d'analyse de conflit (ligne 6). La clause assertive c générée est ajoutée à la base des clauses apprises (ligne 7). L'algorithme effectue donc un retour-arrière au niveau $backJumpLevel + 1$ (ligne 9) et l'apprentissage à partir des succès est seulement appliqué à ce niveau particulier. En restreignant SDCL aux littéraux y propagés au niveau $backJumpLevel + 1$, on assure que chaque nouveau niveau d'implication i pour y est inférieur ou égal au niveau de retour-arrière courant. Un nouveau niveau (appelé $newJumpLevel$) est calculé en choisissant le meilleur niveau (le plus petit) et l'algorithme effectue un retour-arrière à ce niveau (ligne 11). Si la procédure *learnFromSuccess* ne trouve pas de nouvelles implications, $newJumpLevel$ est égale au $backJumpLevel$ (niveau de la clause assertive

classique), et l'algorithme suit le schéma classique d'apprentissage.

La fonction $learnFromSuccess$ est détaillée dans l'algorithme 5. Premièrement, y (resp. U) désigne le littéral de décision (resp. l'ensemble des littéraux propagés à ce niveau) affecté au niveau $currentLevel$ (initialisé à $backJumpLevel + 1$) (ligne 3 et ligne 4). Pour chaque littéral $x \in U$, si on dispose d'une clause assertive binaire de la forme $c = (\beta \vee y \vee x) \in \mathcal{F}$ telle que $\rho(\beta) = faux$, $\rho(x) = \rho(y) = vrai$ est détecté, alors une preuve par résolution générant une clause assertive $\pi_u(x) = \langle \sigma_1, \ldots, \sigma_k \rangle$ est opérée (voir ligne 7). La nouvelle implication σ_k de x est ajoutée à la base des clauses apprises (ligne 8) et le $newJumpLevel$ est mis à jour (ligne 9). Tandis que les littéraux propagés à ce niveau $backJumpLevel+1$ sont traités, le meilleur niveau d'implication ($newJumpLevel$) est retourné (ligne 10), et l'algorithme 3 peut effectuer un retour-arrière à ce nouveau niveau (ligne 11).

Durant le retour-arrière, toutes les décisions entre $backJumpLevel$ et le $newJumpLevel$ sont enregistrées. La fonction $decide()$ les sélectionne en priorité dans le but de maintenir approximativement le même branchement.

Algorithm 5: Learn_from_success

Input: $currentLevel$: niveau d'application de l'apprentissage par succès
Output: $newJumpLevel$: niveau d'implication le plus haut calculé

```
1  begin
2      newJumpLevel = currentLevel-1;
3      y = decisionLiteral(currentLevel);
4      U=UPLiterals(currentLevel);
5      while (∀x ∈ U) do
6          if (∃c = (β ∨ y ∨ x) ∈ F tq. ρ(β) = faux, ρ(x) = ρ(y) = vrai) then
7          Soit π_u(x) = ⟨σ_1, ..., σ_{k-1} = (α ∨ ¬y ∨ x), σ_k⟩ où σ_k = (γ ∨ x) = η[y, σ_{k-1}, c] tq.
           γ = α ∪ β;
8          learn((σ_k);
9          newJumpLevel = min(l(γ),newJumpLevel);
10     end
11     return newJumpLevel;
12 end
```

La figure 3.2 illustre la différence entre un branchement CDCL (à gauche) et SDCL (à droite). Soit $(\alpha \vee \neg d)$ la clause assertive obtenue par l'analyse de conflit classique. Dans l'approche CDCL, l'algorithme effectue un retour-arrière au niveau $backJumpLevel = l(\alpha)$ et affecte le littéral assertif $\neg d$. Dans l'approche SDCL, l'algorithme, effectue un retour-arrière au niveau $backJumpLevel + 1$ et applique le processus d'apprentissage à partir des succès sur tous les littéraux propagés à ce niveau (littéral w dans la figure). Toutes les nouvelles implications pour ces littéraux sont enregistrées dans la base des clauses apprises. Supposons que le meilleur niveau de retour-arrière ($newJumpLevel$) correspond à la nouvelle implication pour w. L'algorithme effectue un retour-arrière au niveau $newJumpLevel$ et affecte le littéral w et les autres littéraux grâce à la mémorisation de l'implication. Finalement, la recherche continue en affectant en priorité les littéraux de décision entre $backJumpLevel$ et $newJumpLevel$ seulement et dans l'ordre tant que c'est possible.

3.4 Expérimentations

Les premières expérimentations ont été menées sur un panel de 286 problèmes industriels issus des deux dernières compétitions SAT-Race'06 et SAT'2007. Toutes les instances testées sont précédées par

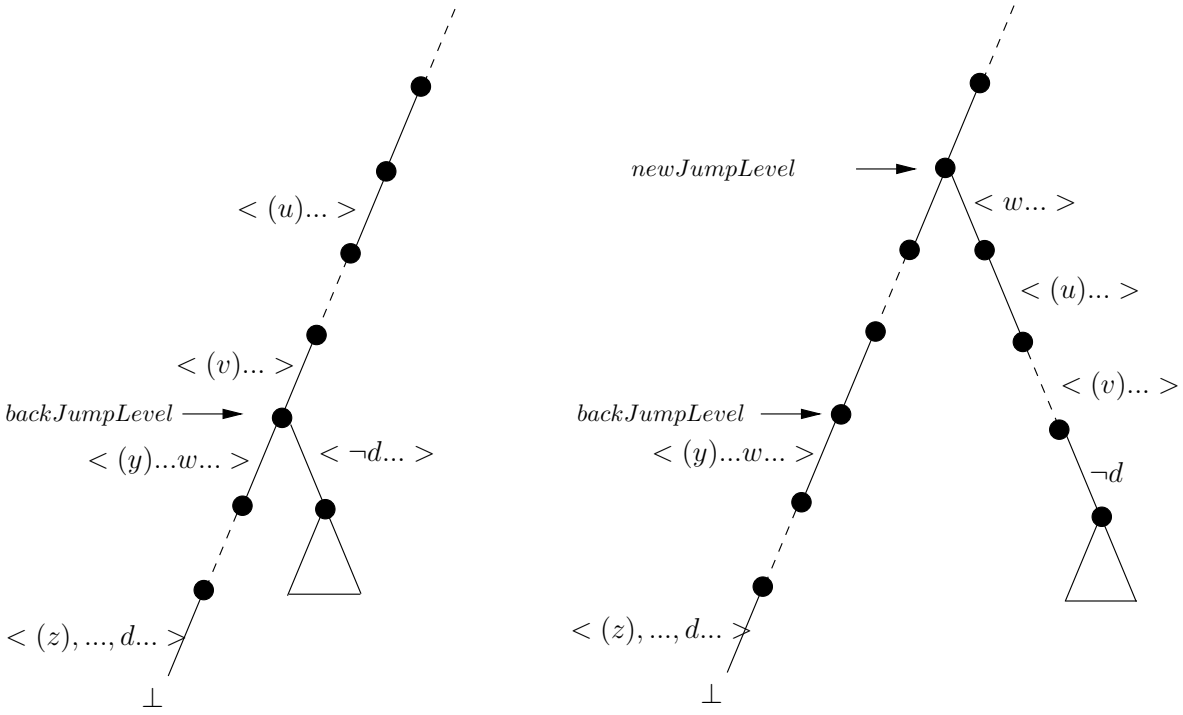


FIG. 3.2 – CDCL Vs SDCL

un pré-traitement *SatELite* [EÉN & BIERE 2005]. Le temps limite de calcul est fixé à 1800 secondes et les résultats sont reportés en secondes. Nous avons implémenté l'extension SDCL (section 3.3) dans MINISAT et nous avons effectué une comparaison entre MINISAT classique et MINISAT modifié par cette approche. Ces tests ont été menés sur un cluster Xeon 3.2GHz (2 GB RAM).

Les figure 3.3 et 3.4 (échelle logarithmique) illustrent une comparaison des résultats de MINISAT et MINISAT +SDCL sur des instances satisfiables et insatisfiables respectivement. L'axe des x (resp. y) correspond à MINISAT +SDCL (resp. MINISAT). Chaque instance est associée à un point de coordonnées (tx, ty) qui représente les temps de calcul obtenus par les deux approches sur l'instance donnée.

La figure 3.3 montre que sur les instances satisfiables, l'apprentissage à partir des succès permet d'accélérer le processus de résolution par rapport à MINISAT. La majorité des points de la figure sont situés sur la diagonale i.e, MINISAT +SDCL est meilleur. Ces résultats ne sont pas surprenants car notre approche visait à corriger le niveau et non pas la valeur de vérité des littéraux impliqués. Ce qui est diffèrent de l'analyse de conflit classique, où on modifie à la fois le niveau d'affectation (niveau assertif) et la valeur de vérité du littéral assertif. D'un autre côté, pour les instances insatisfiables (voir figure 3.4), il n'y a pas de séparation claire entre les performances des deux approches.

Les figures 3.3 et 3.4 (figure du bas) montrent le $temps$ mis par un solveur pour résoudre un certain nombre d'instances. Cette vision globale confirme que notre approche est meilleure sur les instances satisfiables en terme de temps de calcul. Au contraire sur les instances insatisfiables, la différence entre les performances est moins flagrante mais en moyenne MINISAT +SDCL est légèrement meilleur.

Finalement, la table 3.1 résume quelques résultats obtenus par MINISAT +SDCL et MINISAT, pour chaque famille d'instances. Pour chaque famille (première colonne), on reporte le nombre d'instances (deuxième colonne), la moyenne du temps de calcul sur les instances résolues et le nombre d'instances résolues (entre parenthèses). Les résultats sont donnés sur des classes (SAT) et (UNSAT) et sur toutes les instances (SAT-UNSAT). Pour chaque famille, les meilleurs résultats en terme de temps moyen ou de

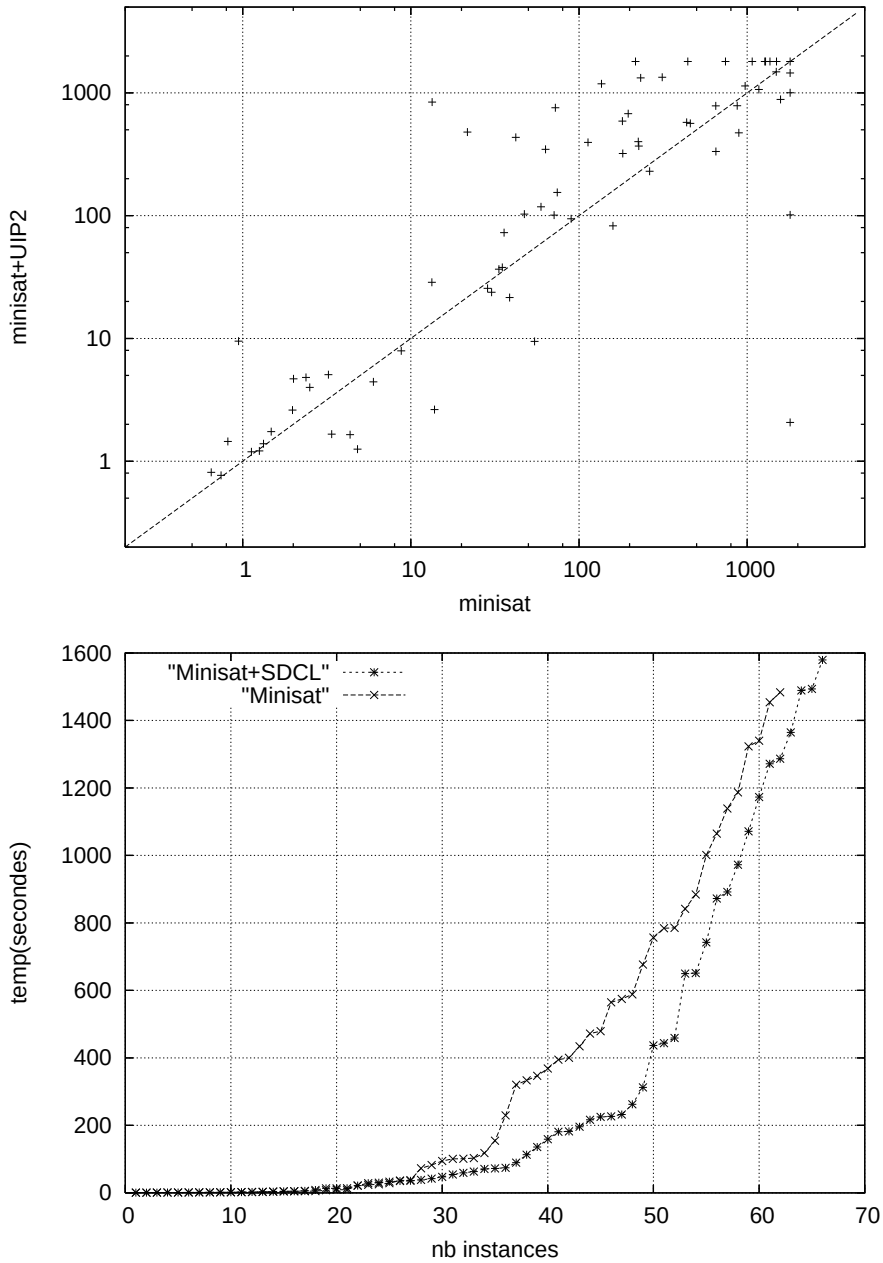


FIG. 3.3 – Instances satisfiables

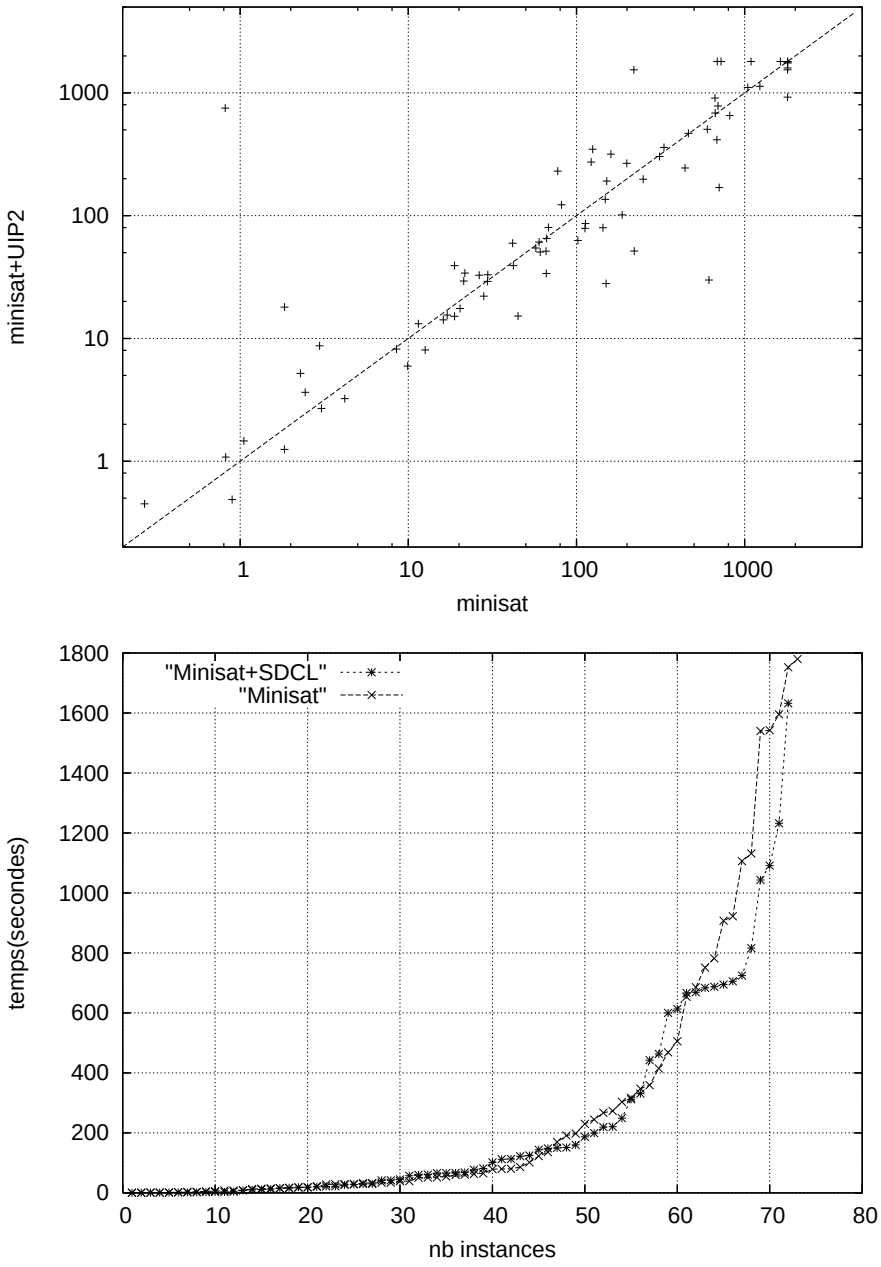


FIG. 3.4 – Instances insatisfiables

		SAT		UNSAT		SAT-UNSAT	
familles	# inst.	Minisat+SDCL	Minisat	Minisat+SDCL	Minisat	Minisat+SDCL	Minisat
mizh_*	10	**501**(10)	720(10)	–	–	**501**(10)	720(10)
manol_*	20	–	–	**336(17)**	344(14)	**336(17)**	344(14)
partial_*	20	**1272(1)**	–	–	–	**1271(1)**	–
total_*	20	262**(7)**	94(4)	66(3)	66(3)	203**(10)**	82(7)
vmp_grieu_*	12	**462**(4)	801(4)	–	–	**462(4)**	801(4)
APro_*	16	13(1)	**2**(1)	323(7)	557**(9)**	284(8)	502**(10)**
dated_*	20	160(9)	**151**(9)	135(2)	661(3)	156(11)	279**(12)**
clause_*	4	**233**(4)	303(4)	–	–	**233**(4)	303(4)
cube_*	4	891(1)	**472**(1)	**59**(1)	60(1)	475(2)	**266**(2)
gold_*	7	–	–	597(4)	**577**(4)	597(4)	**577**(4)
safe_*	4	**13**(1)	841(1)	–	–	**13(1)**	841(1)
ibm_*	20	**98**(10)	121(10)	**36**(9)	102(9)	**69**(19)	112(19)
IBM_*	35	1295**(4)**	1113(3)	**331**(1)	359(1)	1102**(5)**	924(4)
simon_*	6	**149**(2)	277(2)	204(3)	**128**(3)	**182**(5)	188(5)
block_*	2	–	–	27(2)	27(2)	27(2)	27(2)
dspam_*	2	–	–	315(2)	**34**(2)	315(2)	**34**(2)
schup_*	2	**71**(1)	100(1)	**666**(1)	907(1)	**368**(2)	504(2)
sort_*	5	**312**(1)	1339(1)	1232(1)	**1131**(1)	**772**(2)	1235(2)
velev_*	12	–	**2(1)**	25(3)	**19**(3)	25(3)	**15(4)**

TAB. 3.1 – Un aperçu des résultats

nombre d'instances résolues sont mises en gras pour plus de lisibilité. Comme on peut le remarquer sur ces résultats MINISAT +SDCL obtient de meilleurs résultats en général sur des familles satisfiables. Sur la famille $safe-*$ par exemple, SDCL permet un gain de près de deux ordres de grandeur.

Pour résumer, cette première intégration de SDCL sur un solveur moderne est assez prometteuse car elle a mis en lumière une question principale sur l'intérêt de l'ordre d'affectation dans les solveurs modernes. Cette approche est complémentaire au schéma classique d'apprentissage dans la mesure où elle offre une alternative, qui, combinée à CDCL, permet comme cela a été dit de corriger l'interprétation courante.

3.5 Conclusion

Dans cette partie, un nouveau cadre d'apprentissage à partir des succès est proposé. Ce nouveau schéma d'apprentissage permet de dériver de meilleures implications pour la propagation unitaire des littéraux. Ceci peut être vu comme une méthode de réarrangement de l'interprétation courante. Alors que dans les solveurs SAT la partie satisfaite de la formule est ignorée, l'approche suggère que ces clauses connectées au reste de la formule peuvent être exploitées avantageusement durant la recherche. Notre première intégration de SDCL dans un solveur moderne (MINISAT) montre son intérêt à améliorer le temps de calcul sur certaines classes de problèmes industriels. Intéressant encore, on a montré une relation entre la preuve par résolution générant une clause assertive et la déduction basée sur la propagation unitaire. Comme notre approche essaye de réarranger l'interprétation courante, nous envisageons d'approfondir la question du rôle joué par les redémarrages dans ce contexte. Finalement, étendre ce schéma pour récupérer d'autres formes de consistance plus fortes que la propagation unitaire nous semble être une piste de recherche importante à mener pour résoudre des problèmes difficiles.

4

Résolution parallèle de SAT

Sommaire

4.1 Introduction 44
4.2 Stratégies dans les solveurs SAT modernes 45
4.3 Architecture multicore 46
4.4 ManySAT : un solveur SAT parallèle 46
4.4.1 Politique de redémarrage 46
4.4.2 Heuristiques 47
4.4.3 Polarité 47
4.4.4 Apprentissage 47
4.4.5 Partage de clauses 48
4.5 Evaluation 48
4.6 Travaux précédents 48
4.7 Conclusion 50

Dans ce chapitre, un nouveau solveur parallèle ManySAT pour une architecture multicore est présentée. La conception de ManySAT tire parti des principales faiblesses des solveurs SAT modernes, à savoir leurs sensibilités aux réglages des paramètres et leurs manque de robustesse. ManySAT utilise un portfolio complémentaire d'algorithmes séquentiels conçus grâce à une minutieuse variation dans les principales stratégies de résolution. Ces stratégies orthogonales sont combinées au partage de clauses dans le but d'améliorer les performances globales de ManySAT. Ceci contraste avec la majorité des solveurs parallèles existants, qui généralement utilisent le paradigme "diviser pour régner". Les résultats des expérimentations sur plusieurs classes d'instances SAT industrielles, et le premier rang ("gold medal") obtenu par ManySAT lors de la compétition SAT-Race 2008 organisée en Chine, montrent clairement le potentiel qu'il y a derrière la philosophie de conception de notre solveur parallèle.

4.1 Introduction

Ces dernières années, des progrès importants ont été réalisés par les solveurs SAT modernes dans la résolution d'instances industrielles dont la taille peut dépasser des centaines de milliers de variables et des millions de clauses. Cependant, les dernières évaluations (SAT-Race) et compétitions internationales (SAT competition) montrent clairement une stagnation dans l'amélioration des approches existantes. Les gains de performances obtenus peuvent être qualifiés de marginal. Il devient très difficile d'améliorer les performances des solveurs de type Zchaff. De nombreux problèmes industriels restent hors de portée de tous les solveurs existants. Conséquence, de nouvelles approches sont nécessaires pour résoudre ces

instances difficiles. Dans ce contexte, et à la lumière de la prochaine génération d'architectures des ordinateurs, qui dans quatre ou cinq ans peut contenir des dizaine de cores, la conception de solveurs SAT multicore est une excellente perspective de recherche. Cette direction est clairement d'actualité. Une évaluation des solveurs parallèles a été organisée en 2008 (SAT-Race 2008, Chine). Un numéro spécial de la revue internationale sur la satisfiabilité dédié à la résolution parallèle est en cours.

Dans ce chapitre nous détaillons ManySAT, un nouveau solveur parallèle, vainqueur de la dernière compétition Sat-Race 2008 (Track parallèle) [5]. La conception de ManySAT tire avantage de la faiblesse des solveurs SAT modernes : leurs sensibilité aux réglages des paramètres. Pour les instances, changer les paramètres relatifs à la politique de redémarrage (Restart) ou celle du choix de la prochaine variable à affecter peut complètement changer les performances d'un solveur. Dans le contexte de machines multicores, on peut aisément tirer avantage de ce manque de robustesse en construisant un portfolio qui exécute différentes incarnations d'un solveur séquentiel sur la même instance. Chaque solveur contient un paramétrage particulier et leur combinaison devrait représenter un ensemble de stratégies orthogonales et complémentaires.

En outre, les solveurs peuvent échanger des connaissances afin d'améliorer la performance du système au-delà de la performance de chaque solveur individuel. Pour accentuer la robustesse de ManySAT, un partage des clauses apprises est ajouté. Techniquement, ceci est implémenté de manière à minimiser les accès à la base partagée des clauses.

Dans la suite, nous décrivons et nous discutons comment est né ManySAT. Nous donnerons quelques détails des stratégies importantes des solveurs SAT modernes. Nous donnerons ensuite les choix retenus. Des expérimentation sur les instances de la SAT-Race 2008 sont données. Nous discuterons ensuite des travaux connexes en SAT parallèle et nous donnons quelques perspectives.

4.2 Stratégies dans les solveurs SAT modernes

Les solveurs SAT modernes [MOSKEWICZ *et al.* 2001, EÉN & SÖRENSSON 2003] sont basés sur la procédure de recherche DPLL [DAVIS *et al.* 1962] combinée avec (i) politique de redémarrage [GOMES *et al.* 1998, KAUTZ *et al.* 2002], (ii) heuristique de choix de variables (comme VSIDS) [MOSKEWICZ *et al.* 2001], et (iii) l'apprentissage [BAYARDO, JR. & SCHRAG 1997, MARQUES-SILVA & SAKALLAH 1996, MOSKEWICZ *et al.* 2001].

Les solveurs SAT modernes sont basés sur diverses variations dans ces trois importantes stratégies. Il faut noter que ce type de solveurs sont spécialement efficaces sur des instances "structurées" issues d'applications industrielles. Sur ces problèmes, Selman et al. [GOMES *et al.* 2000] ont identifié un phénomène longue traîne (ou **heavy tailed**) i.e, différents ordres de variables souvent mènent à des différences énormes des temps de calcul. Ceci a motivé entre autres l'introduction des stratégies de redémarrage dans les solveurs modernes, qui tentent de découvrir un bon ordre de variables. VSIDS et autres variantes d'heuristiques, pour ne citer que celles-ci, ont été introduites pour éviter le "trashing" et pour diriger la recherche vers la partie la plus contrainte de l'instance. Les redémarrages et VSIDS jouent un rôle complémentaire dans le sens où ils tendent respectivement à diversifier et à intensifier la recherche. L'analyse de conflits "Conflict Driven Clause Learning (CDCL)" est le troisième composant permettant d'effectuer un retour-arrière non-chronologique. À chaque fois qu'un conflit est détecté (au niveau i) une clause assertive est apprise grâce au schéma de résolution basé sur le parcours du graphe d'implication. Ce parcours est utilisé aussi pour mettre à jour l'activité des littéraux mis en cause, permettant à l'heuristique VSIDS de choisir toujours la variable la plus active à chaque point de décision. La clause apprise, appelée clause assertive, est ajoutée à la base des clauses apprises et l'algorithme effectue un retour à un niveau ($j < i$). L'enregistrement de la valeur de vérité ("progress saving") est une autre amélioration

[5] http ://www-sr.informatik.uni-tuebingen.de/sat-race-2008/index.html

intéressante, introduite récemment dans [PIPATSRISAWAT & DARWICHE 2007a] et implémenté dans le solveur Rsat. Il peut être vu comme une nouvelle stratégie de choix de la valeur de vérité (polarité) d'un littéral. Plus précisément à chaque retour-arrière d'un niveau i à un niveau inférieur j, la valeur de vérité des littéraux affectés ou propagés entre ces deux niveaux est mémorisée. Cette polarité est utilisée dans la recherche arborescente pour éviter une résolution répétée des mêmes sous-problèmes. Il est clairement admis qu'une légère variation dans ces stratégies peut conduire à des variations dans les performances des solveurs.

4.3 Architecture multicore

Une architecture multicore peut être vue comme un ensemble de processeurs qui communiquent via une mémoire partagée. En théorie, l'accès à cette mémoire est uniforme, i.e, peut être effectué simultanément. En pratique, l'utilisation des mécanismes de cache dans les unités de calcul crée des problèmes qui peuvent ralentir l'accès à la mémoire.

Notre solveur est construit sur ce modèle de mémoire partagée. La communication entre les solveurs du portfolio est organisée via une structure dédiée où l'opération de lecture peut être faite simultanément, et les mises à jour sont contrôlées par des mécanismes de verrouillage de bas niveau.

4.4 ManySAT : un solveur SAT parallèle

ManySAT est un solveur DPLL incluant toutes les stratégies comme la propagation unitaire, l'heuristique VSIDS, l'analyse de conflits, etc. En plus du schéma classique d'analyse des conflits (schéma First UIP), ManySAT inclut l'extension proposée dans le chapitre 2. Dans la suite, nous décrivons et expliquons certains des choix de stratégies retenus pour ManySAT.

4.4.1 Politique de redémarrage

Notre objectif est d'utiliser des politiques de redémarrage complémentaires pour définir la coupure ("cutoff") du redémarrage x_i i.e, le nombre de conflits maximum à rencontrer avant d'effectuer un redémarrage à la ième exécution. On a décidé d'utiliser différentes politiques de redémarrage : le Luby [LUBY *et al.* 1993b] avec un facteur de 512, une politique géométrique classique $x_i = 1.5 \times x_{i-1}$ avec $x_1 = 100$ [EÉN & SÖRENSSON 2003] et une politique arithmétique lente de la forme $x_i = x_{i-1} + 16000$ avec $x_1 = 16000$.

Pour compléter le portfolio des politiques de redémarrage, une nouvelle politique dynamique a été introduite. Cette politique est basée sur l'évolution de la taille moyenne du retour-arrière. Tout d'abord, une telle information est un bon indicateur de la portion des décisions erronées durant la recherche. Ensuite, elle peut être vue comme une mesure intéressante de la relative difficulté du problème. Notre nouvelle fonction est construite de la façon suivante, pour de grandes (respectivement petites) fluctuations de la taille moyenne des retours-arrière (entre le redémarrage courant et le précédant), il délivre une plus petite (respectivement grande) valeur de la coupure. Elle est définie comme suit : $x_1 = 100, x_2 = 100$, et $x_i = \frac{\alpha}{y_i} \times |cos(1 + r_i)|$, $i > 2$ où $\alpha = 1200$, y_i représente la moyenne de la taille des retours-arrière au redémarrage i, $r_i = \frac{y_{i-1}}{y_i}$ si $y_{i-1} > y_i$, $r_i = \frac{y_i}{y_{i-1}}$ sinon.

Ces politiques sont présentées dans la figure 4.1, où les cores de 0 à 3 implémentent les politiques suivantes respectivement : géométrique, dynamique, arithmétique, et Luby512. Remarquons que la figure utilise une échelle logarithmique sur l'axe des y et la courbe représentant le redémarrage dynamique est obtenue sur l'exécution d'une instance représentative.

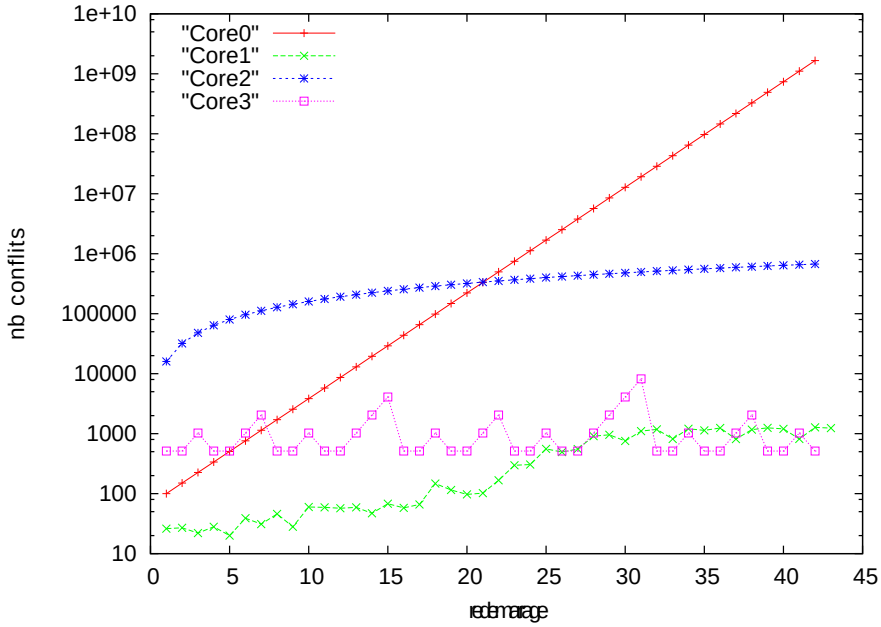


FIG. 4.1 – Stratégies de redémarrage

4.4.2 Heuristiques

Nous avons décidé d'augmenter le niveau du bruit aléatoire associé à l'heuristique VSIDS [MOSKEWICZ *et al.* 2001] du core 0 vu que sa politique de redémarrage est la plus lente. En effet, ce core tend à intensifier la recherche, et augmenter légèrement le bruit aléatoire permet de mieux diversifier la recherche.

4.4.3 Polarité

La polarité du core 0 est liée au nombre d'occurrences d'un littéral dans les clauses apprises. À chaque fois qu'on génère une clause, le nombre d'occurrences de chaque littéral est incrémenté de 1. Pour maintenir une base d'apprentissage plus contrainte, la polarité de l est mise à $vrai$ si $\#occ(l)$ est plus grand que $\#occ(\neg l)$; et à $faux$ sinon. Par exemple, en affectant la polarité de l à $vrai$, on favorise l'apparition des clauses contenant $\neg l$ dans les clauses futures. Cette approche tend à équilibrer la polarité de chaque littéral dans la base des clauses apprises. Ceci a pour effet d'augmenter le nombre de résolvantes possibles entre les clauses de la base des clauses apprises. Comme la politique du core 0 tend à intensifier la recherche, il est important de maintenir une base de clauses de meilleure qualité (plus contrainte). Cependant, pour les politiques de redémarrage rapide, core 1 et core 3, la mémorisation des polarités ("progress saving") est préférable dans le but de sauvegarder les solutions des sous-problèmes indépendants lors des redémarrages ou du retour-arrière. Pour le core 2, une politique statique (affectation toujours à $faux$ d'une variable de décision) est appliquée.

4.4.4 Apprentissage

Les cores 1 et 2 implémentent le schéma classique d'apprentissage basé sur le graphe d'implication. Pour les cores 0 et 3 le graphe d'implication classique est étendu [AUDEMARD *et al.* 2008a] (voir chapitre 2). Comme expliqué dans le chapitre 2, des arcs inverses sont ajoutés au graphe d'implication classique. Ces arcs additionnels sont utilisés pour améliorer le niveau du retour-arrière.

4.4.5 Partage de clauses

Chaque core échange des clauses apprises si leur taille ne dépasse pas une taille limite $e = 16$. Cette limite est obtenue après de nombreux tests exhaustifs visant à déterminer la limite optimale principalement sur des instances industrielles (SAT-Race 2008). La figure 4.2 montre les performances de ManySAT sur 4 cores pour différentes tailles d'échanges e sur un panel d'instances industrielles.

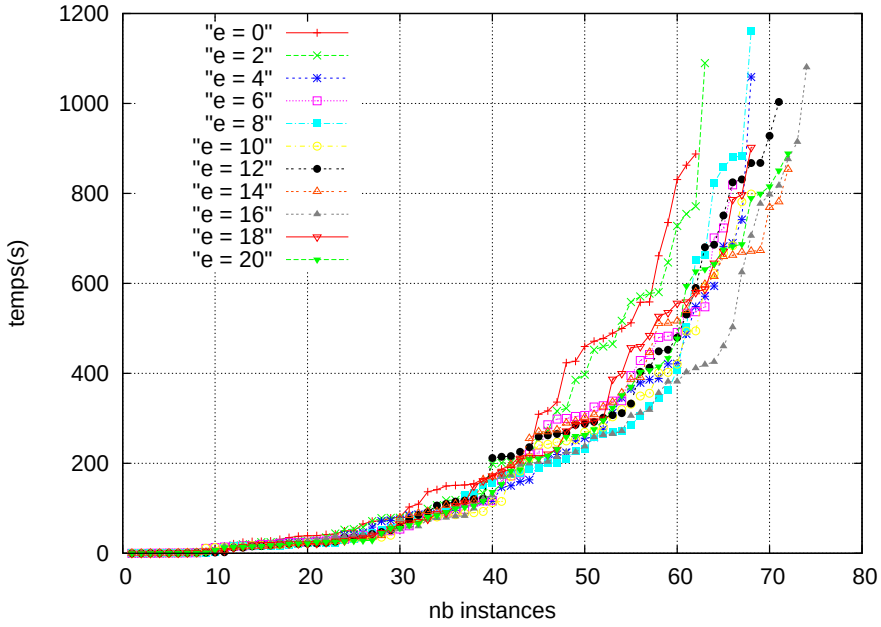


FIG. 4.2 – Différentes limites pour la taille des clauses échangées

La Table 4.1 résume les choix effectués pour les différentes stratégies du solveur parallèle ManySAT.

4.5 Evaluation

ManySAT a été construit au-dessus de minisat 2.02 [EÉN & SÖRENSSON 2003]. SatELite [EÉN & BIERE 2005] est appliqué en pré-traitement. Tous les cores reçoivent donc la même formule simplifiée en entrée.

La figure 4.3 donne les résultats obtenus sur les instances de la SAT race 2008 par ManySAT avec et sans échanges. La valeur limite de la taille des clauses est fixée à 16. L'axe des abscisses donne le nombre d'instances résolues $nb\,instances$ en moins de $temps(s)$ (axe des ordonnées). La figure montre clairement de meilleures performances de notre approche parallèle mais aussi de l'importance de la politique de partage de clauses.

4.6 Travaux précédents

Nous présentons dans cette section les approches les plus connues pour la résolution parallèle d'un problème SAT.

*P*SATO [ZHANG *et al.* 1996] est basé sur le solveur séquentiel SATO (Satisfiability Testing Optimized) [ZHANG & STICKEL 1994]. Comme SATO, il utilise une structure de donnée arborescente pour représenter les clauses. PSATO utilise la notion de chemins de recherche (*guiding-paths*) pour diviser

Stratégies	**Core 0**	**Core 1**	**Core 2**	**Core 3**
Redémarrage	Géométrique $x_1 = 100$ $x_i = 1.5 \times x_{i-1}$	Dynamique (Fast) $x_1 = 100, x_2 = 100$ $x_i = f(y_{i-1}, y_i), i > 2$ si $y_i > y_{i-1}$ $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 + \frac{y_{i-1}}{y_i}) \rvert$ sinon $f(y_{i-1}, y_i) =$ $\frac{\alpha}{y_i} \times \lvert cos(1 + \frac{y_i}{y_{i-1}}) \rvert$ $\alpha = 1200$	Arithmétique $x_1 = 16000$ $x_i = x_{i-1} + 16000$	Luby 512
Heuristique	VSIDS (3% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)
Polarité	if #$occ(l)$ >#$occ(\neg l)$ $l = vrai$ else $l = faux$	Progress saving	faux	Progress saving
Apprentissage	CDCL étendu	CDCL	CDCL	CDCL étendu
partage de clauses	size 16	size 16	size 16	size 16

TAB. 4.1 – ManySAT (SAT-Race 2008)

l'espace de recherche. Ces chemins sont représentés par un ensemble de clauses unitaires ajoutées à la formule originale. L'exploration est organisée sur le modèle maître/esclave. Le maître organise la recherche en adressant les chemins de recherche aux solveurs n'ayant pas d'interactions avec les autres. Le premier qui a fini sa tâche arrête l'exécution du programme. L'équilibrage des tâches entre les solveurs est effectué par le maître.

*P*arallel satz [JURKOWIAK *et al.* 2001] est une parallélisation du solveur séquentiel satz. Il est basé sur le modèle de communication maître/esclave et tend à l'équilibrage de la charge de travail. Le maître partage la charge de travail en attribuant le premier sous-arbre restant du solveur esclave le plus chargé à un solveur esclave libre.

*G*radsat [CHRABAKH & WOLSKI 2003] est basé sur zChaff. Il utilise aussi le modèle maître/esclave et la notion de chemins de recherche pour diviser l'espace de recherche et pour répartir dynamiquement la charge entre les solveurs. Un système de partage de clauses apprises est mis en place. Les clauses d'une taille inférieure à une constante prédéfinie sont partagées. Le solveur intègre une clause quand il effectue un retour au niveau 1 (le plus haut niveau dans l'arbre).

[BLOCHINGER *et al.* 2003] utilise une architecture similaire à Gradsat. Cependant, un client intègre une clause dans sa base si elle n'est pas subsumée par le chemin de recherche courant. En pratique, le partage de clauses est implémenté par des agents mobiles (*mobile-agents*).

*N*agsat [FORMAN & SEGRE 2002] est un solveur SAT parallèle. Il implémente *nagging*, une notion issue du prouveur de théorème DALI. Nagging contient un maître et un ensemble de clients appelés *naggers*. Dans Nagsat, le maître intègre un algorithme DPLL standard avec un ordre de variables statique. Lorsqu'un nagger devient paresseux, il demande un *nagpoint* qui correspond à l'état courant du maître. Lorsqu'il reçoit un nagpoint, il applique une transformation (e.g., un changement d'ordre des variables restantes), et commence sa propre recherche dans le sous-problème correspondant.

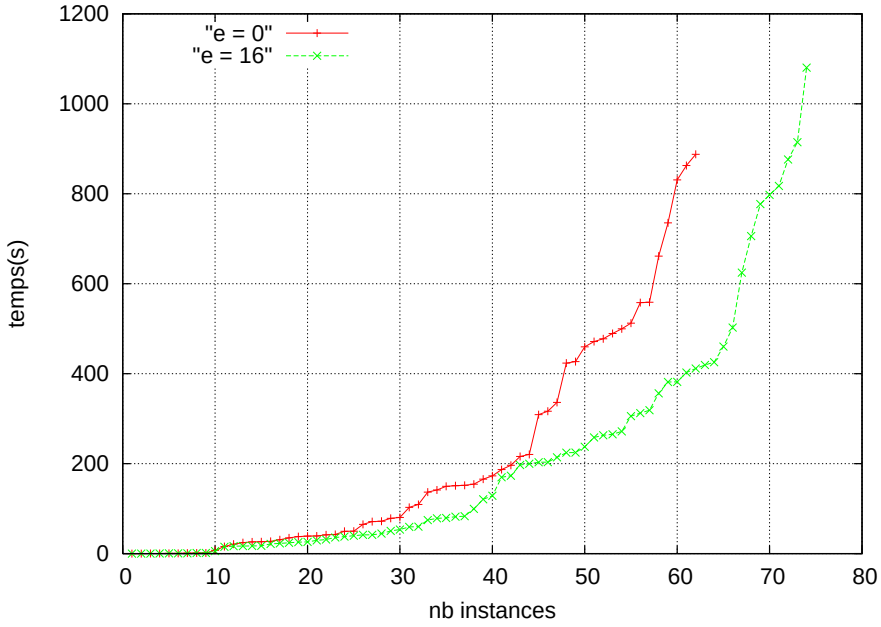


FIG. 4.3 – ManySAT avec et sans échange

Dans [BÖHM & SPECKENMEYER 1996], la formule initiale est divisée dynamiquement en sous-formules disjointes. Chaque sous-formule est résolue par le solveur séquentiel qui tourne sur un processeur particulier. L'algorithme utilise des structures de données optimisées pour modifier les formules booléennes. En outre, l'équilibrage des algorithmes est utilisé pour parvenir à une répartition uniforme de la charge de travail entre les processeurs.

[LEWIS *et al.* 2007] construit pour des architecture multicores à mémoire partagée. Il utilise l'approche "diviser pour régner" où toutes les tâches partagent une unique formule représentant la formule originale augmentée par les clauses apprises. Lorsqu'une clause est apprise par une tâche, il utilise un verrou pour permettre une mise à jour de la base commune.

[CHU & STUCKEY 2008] utilise une approche classique de "diviser pour régner" basée sur les chemins de recherche. Cependant, il exploite les connaissances sur ces chemins pour améliorer la qualité des clauses apprises. En effet, les clauses peuvent être de grandes tailles, mais en prenant en considération la connaissance du chemin de recherche d'une tâche particulière, une clause peut devenir plus petite et par conséquent plus intéressante. Ceci permet à $pMinsat$ d'étendre le partage de clauses du moment que les clauses de grandes tailles deviennent petites dans le contexte d'une autre recherche.

4.7 Conclusion

Nous avons présenté ManySAT, un solveur SAT parallèle qui exploite avantageusement les architectures multicores. ManySAT est fondé sur une compréhension de la principale faiblesse des solveurs SAT modernes séquentiels liée à leur sensibilité aux réglages des paramètres. Comme résultat, ManySAT utilise un portfolio d'algorithmes séquentiels et complémentaires qui coopèrent pour améliorer la performance globale du solveur. La conception de ManySAT nous a beaucoup appris sur la combinaison de stratégies (non)similaires dans un portfolio. Dans le futur nous étudierons cet aspect d'un point de vue

théorique et pratique afin d'améliorer la coopération dans la résolution parallèle de SAT.

5

Graphe SAT : une nouvelle forme de représentation

Sommaire

5.1 Introduction 53
5.2 Graphe-SAT : une nouvelle représentation de formules CNF 53
5.3 Pontages algorithmiques : CNF / Graphe-SAT 55
5.3.1 Résolution & Fermeture Transitive 55
5.4 Graphe-SAT : Applications 57
5.4.1 Ensembles 2-SAT strong backdoors 57
5.4.2 Génération d'instances SAT difficiles 58
5.4.3 Graphe & pré-traitement 62
5.5 Expérimentations 64
5.5.1 Calcul d'ensembles 2-SAT Strong Backdoors 64
5.5.2 Évaluation du Générateur Diamond 67
5.5.3 Évaluation du pré-traitement 69
5.6 Conclusion 73

Nous proposons dans ce chapitre une nouvelle représentation sous forme de graphe d'une formule CNF [AUDEMARD *et al.* 2008b]. Elle étend le graphe d'implication généralement utilisé dans le cas des clauses binaires (2-SAT) au cas général. Chaque clause est représentée comme un ensemble d'implications conditionnelles. Dans notre cas, par implication conditionnel on exprime l'assertion suivante : un littéral a implique b sous la condition C où C est une conjonction de littéraux. Dans le cas 2-SAT, C est un ensemble vide. Cette notion d'implication conditionnelle nous permet pour des clauses de tailles quelconques de rester dans le cadre d'une représentation en termes de graphes valués, au lieu d'une représentation en hyper-graphes, généralement plus difficile à manipuler. Une clause est donc exprimée par une implication conditionnelle qui est elle même codée par un arc étiqueté par un ensemble de littéraux, appelé *contexte, ou condition*. Cette nouvelle représentation permet d'étendre quelques propriétés algorithmiques intéressantes utilisées dans le cade du fragment traitable 2-SAT (voir la section 5.4.1). Parmi elles, la résolution classique est reformulée en utilisant la fermeture transitive d'un graphe. Les (plus courts) chemins entre les nœuds $\neg a$ et a donnent une façon originale de calculer les conditions (minimales) sous lesquelles un littéral a est impliqué. Cette nouvelle représentation offre de nombreux avantages et diverses utilisations intéressantes. Dans le cadre de ce chapitre, trois utilisations concrètes de cette représentation de formules CNF quelconques en graphe sont présentées et validées expérimentalement. La première concerne l'extraction des ensembles 2-SAT strong backdoors. Dans la seconde, nous

exploitons cette représentation pour la génération d'instances SAT difficiles. Alors que dans la troisième, nous dérivons une nouvelle technique de pré-traitement des formules CNF.

5.1 Introduction

Il est bien connu que l'efficacité des algorithmes de résolution est dépendante de la représentation utilisée. En d'autres termes, un calcul ou une opération, même s'il est de complexité polynomiale, d'un point de vue pratique, peut présenter des différences notables en termes d'efficacité. Il est donc important de bien choisir la représentation qui réduit au maximum la complexité des opérations les plus fréquemment utilisées. Un compromis est donc à trouver entre la représentation et l'efficacité des opérations associées. Dans le cadre SAT, la formule booléenne générale exprimant un problème donné est traditionnellement transformée (avec l'ajout de variables supplémentaires) sous forme normale conjonctive (CNF) équivalente pour la satisfaisabilité. Cependant, le codage sous forme CNF induit une perte des connaissances structurelles qui sont mieux exprimées dans d'autres formalismes et qui peuvent permettre une résolution plus efficace [KAUTZ & SELMAN 1997, GREGOIRE *et al.* 2005, THIFFAULT *et al.* 2004]. Pour exploiter ces connaissances structurelles, des travaux récents ont été proposés. Quelques uns utilisent la forme étendue des formules booléennes (non CNF [THIFFAULT *et al.* 2004], contraintes pseudo booléennes [DIXON & GINSBERG 2002]) pour le codage des formules. Tandis que d'autres essayent de récupérer et/ou déduire des propriétés structurelles à partir des formules CNF (symétries [ALOUL *et al.* 2003a], dépendances fonctionnelles [GREGOIRE *et al.* 2005], équivalences [LI 2003]). Finalement, différents graphes ont été élaborés pour présenter ou modéliser ou bien même résoudre des instances SAT. Ce dernier type d'approche est motivé par la visualisation de structures [SINZ & DIERINGER 2005], la décomposition [DARWICHE 2004], la propagation [LU *et al.* 2003], l'apprentissage (à partir du graphe d'implication) [MARQUES-SILVA & SAKALLAH 1996, MOSKEWICZ *et al.* 2001].

Dans ce chapitre, un nouvelle représentation en graphe est proposée. Elle étend de manière originale le graphe d'implication pour les formules binaires (2-SAT) au cas général. Comme indiqué au début de ce chapitre, chaque clause est représentée par un ensemble d'implications (conditionnelles) et codée avec différents arcs étiquetés avec un ensemble de littéraux. C'est une extension naturelle de la représentation en graphes des clauses binaires. Au-delà de l'étude (ou de l'extension) de certaines opérations classiques (e.g. preuve par résolution) du cas CNF aux graphes (e.g. fermeture transitive du graphe), notre but est de montrer d'autres utilisations où cette nouvelle représentation est clairement appropriée.

Dans la suite de ce chapitre, après la définition de notre nouvelle représentation d'une formule CNF en graphes, une description formelle de ses caractéristiques est détaillée. Plus particulièrement la résolution est reformulée en utilisant la fermeture transitive du graphe. Trois utilisations pratiques de cette représentation sont ensuite proposées. La première concerne le calcul des ensembles strong backdoors, la seconde concerne l'utilisation de ce graphe pour générer des instances SAT difficiles et finalement une technique de pré-traitement des formules CNF. Nous terminons par une étude expérimentale menée sur des instances issues des dernières compétitions SAT.

5.2 Graphe-SAT : une nouvelle représentation de formules CNF

Comme mentionné dans l'introduction, notre représentation peut être vue comme une extension du graphe d'implication souvent utilisé pour représenter des formules 2-SAT [ASPVALL *et al.* 1979] au cas général.

Avant d'introduire notre approche, nous rappelons cette représentation en graphe dans le cas des clauses binaires.

Définition 39 (Représentation Graphe-2-SAT) *Soit $\mathcal{F}$ une formule 2-SAT. Le graphe associé à $\mathcal{F}$, est défini comme $G_{\mathcal{F}} = (S, E)$, où*
- $S = \{x, \neg x | x \in \mathcal{L}(\mathcal{F})\}$
- $E = \{(\neg x, y), (\neg y, x) | (x \vee y) \in \mathcal{F}\}$

L'algorithme polynomial utilisé pour déterminer la satisfaisabilité d'une formule 2-SAT $\mathcal{F}$ est basé sur l'application de la fermeture transitive de $G_{\mathcal{F}}$, $tr(G_{\mathcal{F}}) = (S, E')$ et sur la vérification de l'existence ou non d'un littéral $x \in S$ tel que $(x, \neg x) \in E'$ et $(\neg x, x) \in E'$. Si un tel littéral existe alors $\mathcal{F}$ est insatisfiable sinon $\mathcal{F}$ est satisfiable.

Évidemment, le calcul de la fermeture transitive du graphe $G_{\mathcal{F}}$ est équivalent à la saturation de $\mathcal{F}$ par résolution. En effet, pour (x, y) et (y, z) de E un nouvel arc (x, z) est généré et ajouté au graphe. Une telle application correspond à $res(y, (\neg x \vee y), (\neg y \vee z)) = (\neg x \vee z)$.

Pour une formule CNF quelconque, une représentation naturelle peut être obtenue en utilisant un hyper-graphe où les nœuds sont représentés par les littéraux et, les hyper-arêtes correspondent aux clauses. L'inconvénient de cette approche réside dans la difficulté à traiter les hyper-graphes.

Dans ce qui suit, nous généralisons le graphe d'implication des formules 2-SAT pour des formules propositionnelles quelconques. Cette généralisation est basée sur la notion d'implication conditionnelle.

Définition 40 *Soit c une clause telle que $|c| \geq 2$. Un contexte (ou condition) η_c associé à c est une conjonction de littéraux telle que $\neg\eta_c \subset c$ et $|c - \{\neg\eta_c\}| = 2$, i.e, quand le contexte η_c est vrai, la clause c devient une clause binaire.*

Exemple 7 *Soit $c = (a \vee \neg b \vee c \vee d)$ une clause. Un contexte possible associé à c est $\eta_c = (b \wedge \neg c)$. La clause c peut être réécrite comme $((\neg a \wedge \eta_c) \rightarrow d)$.*

Pour une clause c de taille k, on a $|\eta_c| = k - 2$. Le contexte associé à une clause binaire est vide, tandis que pour les clauses ternaires, le contexte est réduit à un seul littéral. Le nombre des contextes possibles de c est égal à $\frac{k(k-1)}{2}$. Une clause c peut être réécrite en $k(k-1)$ différentes façons. Quand $k = 1$, la clause est unitaire, aucun contexte n'est possible.

En utilisant la clause c de l'exemple 7, on obtient six contextes possibles de taille 2 et douze réécritures différentes de c.
Définissons maintenant le graphe représentatif d'une formule CNF.

Définition 41 (Représentation Graphe-SAT) *Soit $\mathcal{F}$ une formule CNF. Nous définissons $G_{\mathcal{F}} = (S, E, v)$ comme étant le graphe SAT associé à $\mathcal{F}$ défini comme suit*
- $S = \{x, \neg x | x \in \mathcal{L}(\mathcal{F})\}$
- $E = \{a = (\neg x, y, v(a)) | \exists c \in \mathcal{F}, c = (x \vee \neg\eta_c \vee y)$ *et* $v(a) = \eta_c\}$

Exemple 8 *Soit la CNF suivante :*

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

La figure 5.1, montre le graphe $G_{\mathcal{F}}$ de la formule $\mathcal{F}$

Dans la suite, pour des raisons de clarté, on note parfois l'arc étiqueté $a = (x, y, v(a))$ comme $a = (x, y)$. On note aussi $cla(a) = (\neg x \vee \neg v(a) \vee y)$ comme étant la clause associée à a.

De manière évidente, la définition 41 généralise le graphe classique 2-SAT. En effet, si tous les contextes sont vides, i.e, toutes les clauses de $\mathcal{F}$ sont binaires, alors tous les arcs de $G_{\mathcal{F}}$ sont étiquetés avec un ensemble vide de littéraux.

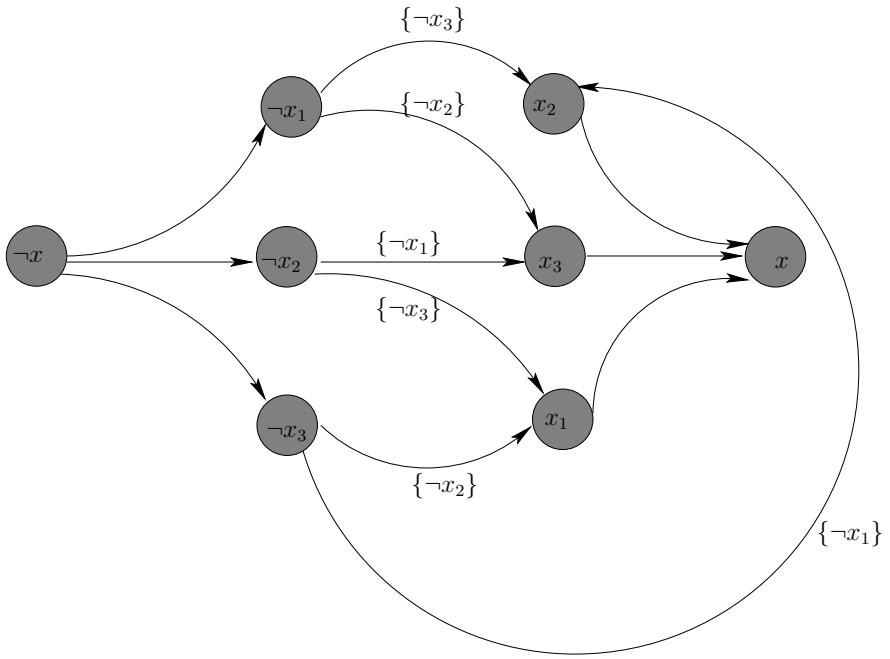


FIG. 5.1 – Graphe SAT $G_{\mathcal{F}}$ associé la formule $\mathcal{F}$

5.3 Pontages algorithmiques : CNF / Graphe-SAT

5.3.1 Résolution & Fermeture Transitive

Dans cette section, nous démontrons l'équivalence entre la résolution classique et la fermeture transitive du graphe. Nous commençons par introduire quelques définitions nécessaires et nous considérons dorénavant uniquement les formules $\mathcal{F}$ sans clauses tautologiques. De plus, $G_{\mathcal{F}} = (S, A, v)$ indique le graphe associé à $\mathcal{F}$.

Définition 42 *Soit $G_{\mathcal{F}} = (S, A, v)$ un graphe, $a_1 = (x, y, v(a_1)) \in A$ et $a_2 = (y, z, v(a_2)) \in A$. Nous définissons $tr(a_1, a_2) = a_3$ tel que $a_3 = (x, z, v(a_1) \cup v(a_2) \backslash \{x, \neg z\})$.*

Dans la définition ci-dessus, l'élimination de $\{x, \neg z\}$ du contexte associé à a_3 garantit que la clause $cla(a_3)$ ne contient pas plusieurs occurrences du même littéral. Ce qui correspond à l'application de la règle de fusion.

Exemple 9 *Soit $a_1 = (x, y, \{\neg z, e\})$ et $a_2 = (y, z, \{x, f\})$. Les deux clauses codées respectivement par a_1 et a_2 sont $c_1 = (\neg x \vee z \vee \neg e \vee y)$ et $c_2 = (\neg y \vee \neg x \vee \neg f \vee z)$. On obtient $tr(a_1, a_2) = (x, z, \{\neg e, \neg f\})$ et $cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \neg f \vee z)$. Cette dernière clause ne contient pas de littéraux redondants. En utilisant la résolution, $res(c_1, c_2, y)$ on obtient donc $c_3 = (\neg x \vee z \vee \neg e \vee \neg x \vee \neg f \vee z)$.*

En appliquant la règle de fusion sur c_3, on élimine une occurrence de $\neg x$ et z. On obtient $res(c_1, c_2, y) = cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \neg f \vee z)$

Propriété 10 *Soit $c_1 = (x \vee \eta_{c_1} \vee y) \in \mathcal{F}$, $c_2 = (\neg y \vee \eta_{c_2} \vee z) \in \mathcal{F}$, $a_1 = (x, y, \eta_{c_1}) \in A$ et $a_2 = (x, y, \eta_{c_2}) \in A$. Nous avons $res(c_1, c_2, y) = cla(tr(a_1, a_2))$*

La preuve de la propriété 10 est une conséquence directe des définitions 41 et 42.

Définition 43 (chemin) *Un chemin $p(x, y)$ entre x et y dans $G_{\mathcal{F}}$, est défini comme suit : $p(x, y) = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$ tel que $x_{i_1} = x$ et $x_{i_k} = y$ et $1 < j \leq k$ $(x_{i_{j-1}}, x_{i_j}) \in A$.*
Nous définissons $\eta_p = \bigcup_{1<j\leq k} v((x_{i_{j-1}}, x_{i_j}))$ le contexte associé à $p(x, y)$
et $tr(p(x, y)) = tr(\ldots(tr((x_{i_1}, x_{i_2}), (x_{i_2}, x_{i_3}))\ldots(x_{i_{k-1}}, x_{i_k})\ldots)$,
comme étant la fermeture transitive associée à $p(x, y)$.

Définition 44 (chemin fondamental) *Soit $p(x, y) = [x = x_{i_1}, x_{i_2}, \ldots, x_{i_k} = y]$ un chemin entre x et y. $p(x, y)$ est dit fondamental ssi il satisfait les deux conditions suivantes :*

1. *η_p ne contient pas un littéral et son opposé.*
2. *$\neg x \notin \eta_p$ et $y \notin \eta_p$*

La propriété suivante montre qu'à partir d'un chemin fondamental $p(x, y)$ on peut dériver une résolvante fondamentale en utilisant la fermeture transitive.

Propriété 11 *Le chemin $p(x, y)$ est fondamental ssi $cla(tr(p(x, y)))$ est une clause fondamentale.*

Preuve : Pour un chemin fondamental $p(x, y)$, η_p ne contient pas deux littéraux opposés. Comme $cla(tr(p(x, y))$ peut être réécrite sous forme $r = (\neg x \vee \neg\eta_p \vee y)$, r est clairement une clause fondamentale. En effet, d'après la définition 44, η_p ne contient pas un littéral et son opposé (première condition) et $x \in \neg\eta_p$ et $\neg y \notin \neg\eta_p$ (seconde condition). L'inverse est aussi vrai. Supposons que $cla(tr(p(x, y)))$ n'est pas fondamentale. Comme $tr(p(x, y)) = (x, y, \eta_p)$, la clause $cla(tr(p(x, y)) = (\neg x \vee \neg\eta_p \vee y)$ n'est pas fondamentale. Ce qui montre que la première ou la deuxième condition de la définition 44 n'est pas satisfaite.

Dans ce qui suit, nous décrivons comment la résolution classique peut être appliquée en utilisant la fermeture transitive.

Définition 45 *Soit $G_{\mathcal{F}} = (S, A, v)$ le graphe associé à $\mathcal{F}$. Nous définissons $tr(G_{\mathcal{F}})$ comme le graphe (S, A', v') tel que : $\forall x \in S$, $\forall y \in S$ tel que $\exists p(x, y)$ un chemin fondamental entre x et y dans $G_{\mathcal{F}}$, $A' = A \cup \{tr(p(x, y))\}$.*

Définition 46 *Nous définissons la fermeture transitive d'un graphe $G_{\mathcal{F}}$, notée $G^{ft}_{\mathcal{F}}$, comme le graphe $tr^k(G_{\mathcal{F}})$ tel que $tr^k(G_{\mathcal{F}}) = tr^{k+1}(G_{\mathcal{F}})$ où tr^k représente k fois l'application de tr.*

Il est important de noter que dans le cas 2-SAT, une seule application de la transitivité est suffisante pour atteindre le point fixe (un graphe fermé par transitivité). En introduisant, les valuations ensemblistes sur les arcs, la fermeture transitive qui est polynomiale dans le cas 2-SAT devient exponentielle dans le cas général.

Propriété 12 *Soit $\mathcal{F}$ une formule CNF. $\mathcal{F}$ est insatisfiable ssi $\exists k$ tel que $tr^k(G_{\mathcal{F}}) = (S, A', v')$ et $\exists x \in S$ tel que $(x, \neg x, \emptyset) \in A'$ et $(\neg x, x, \emptyset) \in A'$*

Preuve : Notre graphe est complet, i.e, chaque clause est codée $k(k-1)$ fois. Comme montré précédemment, l'application de tr sur les arcs de $G_{\mathcal{F}}$ est équivalente à l'application de la résolution entre les clauses de $\mathcal{F}$. La preuve peut être dérivée de la preuve de la complétude pour la réfutation de la résolution classique.

Dans la propriété 12, nous avons montré comment appliquer la résolution classique sur le graphe. Évidemment, d'autres techniques SAT (e.g. élimination de variable, propagation unitaire) peuvent être reformulées en utilisant ce même graphe. Notons que ce graphe peut avoir plusieurs caractéristiques intéressantes. L'un des principaux avantages est que la dépendance entre les clauses est mieux exprimée

en utilisant une telle représentation. Une telle propriété structurelle est connue pour être déterminante dans l'efficacité des solveurs SAT. Plus intéressant encore, le graphe SAT peut être vu comme un graphe d'états, où chaque état représente un littéral et une transition entre les deux états s_1 et s_2 est représentée par une condition $v((s_1, s_2))$ (un ensemble de littéraux). Plusieurs problèmes intéressants peuvent être reformulés plus facilement. Par exemple, le calcul du plus court chemin entre un littéral et son opposé, i.e, le calcul des conditions minimales pour qu'un littéral puisse être impliqué.

Dans la section suivante, trois exploitations possibles de notre graphe sont décrites.

5.4 Graphe-SAT : Applications

Comme mentionné dans l'introduction, différentes représentations sous forme de graphe d'une formule CNF ont été proposées dans la littérature. Elles ont été introduites pour différentes utilisations, comme par exemple pour apprendre des clauses dans les solveurs SAT de type CDCL (voir chapitre 1), simplifier les formules CNF [BRAFMAN 2001], résoudre des instances aléatoires 3-SAT (survey propagation) [BRAUNSTEIN *et al.* 2005] et la visualisation de la structure des formules CNF [SINZ & DIERINGER 2005].

Dans la définition de $G_{\mathcal{F}}$, chaque clause est représentée $k(k-1)$ fois. Une telle représentation multiple d'une clause donnée est nécessaire pour que la fermeture transitive soit complète pour la réfutation. Pour des formules de grandes tailles, la représentation complète du graphe est impraticable.

Suivant à quelle fin ce graphe est utilisé, des restrictions utiles peuvent être définies. Par exemple, si on représente chaque clause avec seulement un seul arc, le graphe obtenu est équivalent à la formule CNF originale.

Nous devons seulement nous rappeler que dans ce cas certaines résolvantes pouvant être obtenues en employant la fermeture transitive du graphe ne peuvent plus être dérivées à partir de ce graphe restreint.

5.4.1 Ensembles 2-SAT strong backdoors

Dans cette section, nous allons montrer comment la représentation sous forme de graphe peut être utilisée pour le calcul des ensembles strong backdoors.

Par ces connexions à la difficulté des instances SAT, la notion de (strong) backdoors introduite par Williams-et al dans [WILLIAMS *et al.* 2003] a suscité depuis un réel intérêt. Un ensemble de variables forme un *backdoor* d'une formule, si il existe une affectation de ces variables rendant la formule simplifiée traitable en temps polynomial. Un tel ensemble de variables est appelé *strong backdoors* si, pour toute affectation de ces variables, le reste de la sous-formule est traitable polynomialement. Ce type de structure est relié à la notion de variables indépendantes [KAUTZ *et al.* 1997, GIUNCHIGLIA *et al.* 2002] et à l'ensemble coupe cycle introduit pour le problème de satisfaction de contraintes [DECHTER 1989].

Rappelons que le calcul de l'ensemble strong backdoors minimal est un problème NP-difficile. En pratique, une approximation d'un strong backdoors de taille "*raisonnable*" constitue un vrai challenge à l'heure actuelle.

Des travaux précédents ont été effectués dans ce but. Par exemple, l'approche proposée dans [GREGOIRE *et al.* 2005] essaye d'extraire un ensemble de portes (fonctions booléennes) à partir d'une formule CNF donnée et, en exploitant les dépendances entre les variables, un strong backdoors est calculé. Le principal inconvénient de cette approche est que de nombreux problèmes ne contiennent pas de telles propriét$\mathcal{F}^m$és structurelles (fonctions booléennes). Récemment, dans [MAAREN & NORDEN 2005] un concept de fraction reverse Horn d'une formule CNF est proposé et une importante corrélation entre la densité des instances aléatoires 3-SAT et les performances des solveurs SAT est mise en évidence. Dans [LYNCE & MARQUES-SILVA 2004], la relation entre la complexité des instances aléatoires 3-SAT insa-

tisfiables et les ensembles strong backdoors est étudiée. Une autre approche a été proposée récemment par Paris et al.[PARIS *et al.* 2006]. Elle est basée sur le calcul d'un renommage permettant de maximiser la taille de la sous-formule de Horn. Une approximation est obtenue en adaptant une méthode de recherche locale pour calculer le meilleur renommage. Le strong backdoors est ensuite calculé à partir de la sous-formule non Horn. Les ensembles strong backdoors obtenus sont appelés Horn strong backdoors.

Comme notre représentation sous forme de graphe est une généralisation du graphe d'implication 2-SAT, les ensembles strong backdoors que nous proposons de calculer sont considérés par rapport au fragment polynomial 2-SAT, i.e, l'ensemble de variables B tel que pour toute affectation ρ des variables de B, la formule simplifiée par ρ appartient au fragment polynomial 2-SAT.
Notons que pour calculer les ensembles 2-SAT strong backdoors, il suffit de considérer une restriction du graphe. Chaque clause de $\mathcal{F}$ est représentée avec seulement un seul arc dans $G_{\mathcal{F}}$. On note $G^u_{\mathcal{F}}$ le graphe restreint associé à $\mathcal{F}$, appelé graphe unique.
La propriété suivante montre comment on peut utiliser le graphe unique $G^u_{\mathcal{F}}$ pour calculer les ensembles 2-SAT strong backdoors.

Propriété 13 *Soit $\mathcal{F}$ une formule et, $G^u_{\mathcal{F}} = (S, A, v)$ son graphe associé.*
$B = \cup_{a_i \in A}\{\mathcal{V}(l) | l \in v(a_i)\}$ est un ensemble 2-SAT strong backdoors.

Preuve : Pour prouver cette propriété, il suffit de montrer que pour toute affectation ρ de B, $\mathcal{F}|_\rho$ est une formule 2-SAT. Chaque arc $a_i = (x, y, v(a_i)) \in A$ code une clause $c = (\neg x \vee \neg v(a_i) \vee y) \in \mathcal{F}$. Pour chaque clause $c \in \mathcal{F}$, toutes les variables de $v(a_i)$ apparaissent dans B. Par conséquent, soit ρ satisfait au moins un littéral dans $\neg v(a_i)$, soit tous les littéraux dans $\neg v(a_i)$ sont faux. Ainsi, la formule est soit 2-SAT, non satisfaite ou satisfaite. B est donc un 2-SAT strong backdoors.

Rappelons que notre objectif principal est l'approximation de l'ensemble strong backdoors 2-SAT minimal. Pour arriver à cette fin, on utilise $G^u_{\mathcal{F}}$ le graphe unique de $\mathcal{F}$. De plus, comme une clause c de taille k peut être codée de plusieurs façons, nous avons besoin de choisir un arc parmi les $k(k-1)$ possibles. Pour construire $G^u_{\mathcal{F}}$, à chaque étape une nouvelle clause $c = (x \vee \eta_c \vee y) \in \mathcal{F}$ de taille k est traitée et un nouvel arc $a = (x, y, v(a))$ où $v(a) = \neg\eta_c$, est ajouté au graphe. Premièrement, tous les littéraux de c appartenant aux précédents contextes sont inclus dans $v(a)$. Deuxièmement, les littéraux x et y sont choisis parmi les littéraux restants (ceux qui apparaissent le moins dans $\mathcal{F}$). De plus, la propriété suivante est utilisée pour réduire la taille de l'ensemble 2-SAT strong backdoors.

Propriété 14 *Soient B un ensemble 2-SAT strong backdoors d'une formule CNF $\mathcal{F}$ et $b \in B$. Si $\forall c \in \mathcal{F}$ tel que $b \in c$ ou $\neg b \in c$, $|\mathcal{V}(c) \cap B| > |c| - 2$, alors $B - \{b\}$ est un 2-SAT strong backdoors.*

La preuve de la propriété 14 est évidente. Cette dernière exprime le fait qu'une variable de l'ensemble 2-SAT strong backdoors peut être éliminée si elle apparaît seulement dans les clauses avec au plus un littéral n'appartenant pas à B, i.e, l'élimination de telles variables aboutit à des clauses contenant au plus deux littéraux qui ne sont pas dans B.

L'algorithme 6 décrit notre approche pour le calcul du 2-SAT strong backdoors. Comme nous cherchons des ensembles 2-SAT strong backdoors, toutes les clauses binaires de $\mathcal{F}$ sont supprimées (ligne 2) et, en utilisant l'heuristique décrite précédemment, le graphe $G^u_{\mathcal{F}}$ est construit sur l'ensemble des clauses restantes. Un premier ensemble B est calculé (ligne 4) et réduit en utilisant la propriété 5 (ligne 5 à 10).

5.4.2 Génération d'instances SAT difficiles

Dans cette section, nous montrons comment utiliser la représentation Graphe-SAT pour générer des instances difficiles . Notre construction est basée sur la traversée du chemin $p(a, \neg a)$ à partir d'un littéral a vers son opposé $\neg a$. Notons que lorsque l'ensemble des contextes cumulés le long de p est vide,

Algorithm 6: Approximation d'un ensemble 2-SAT strong backdoors

Input: une formule CNF $\mathcal{F}$
Output: un ensemble 2-SAT strong backdoors B

```
1  begin
2      Supprimer les clauses binaires de F
3      Créer le graphe G^u_F = (S, E)
4      B = ⋃_{a∈E} V(v(a))
5      foreach c ∈ F do nb(c) = |{x ∉ c ∩ B}|
6      foreach u ∈ B do
7          bool remove=vrai
8          foreach (c ∈ F | u ∈ c or ¬u ∈ c) do
9              if (nb(c) = 2) then remove = faux
10         end
11         if remove then B = B − {u}
12     end
13 end
```

alors $\neg a$ est un littéral impliqué, i.e, $cla(tr(p(a, \neg a))) = \neg a$. Cependant, décider de l'implication d'un littéral ou trouver l'ensemble de conditions minimales pour son implication est un problème difficile. Pour générer des instances SAT difficiles, notre idée consiste à construire un graphe particulier codant une instance CNF où l'implication de $\neg a$ est triviale à obtenir sur la représentation Graphe-SAT mais difficile à calculer pour les solveurs SAT. Le générateur que nous proposons combine de façon originale la représentation sous forme de graphe et une règle de résolution très connue appelée l'*hyper-résolution*.

Définition 47 *Soit $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ une clause, appelée side clause et $\mathcal{F}_h$ une formule CNF $(\neg y_1 \vee \alpha) \wedge (\neg y_2 \vee \alpha) \wedge \cdots \wedge (\neg y_m \vee \alpha)$, où α est une sous-clause. En appliquant l'hyper-résolution sur c_s et $\mathcal{F}_h$, qu'on notera hr(c_s, $\mathcal{F}_h$), on peut dériver la sous-clause α, appelée hyper-résolvante. La formule $\mathcal{F}_h \cap c_s$ est appelée une hyper-formule.*

L'hyper-résolution peut être définie en utilisant la résolution classique. En effet, α peut aussi être dérivée de c_s et $\mathcal{F}_h$ en utilisant m différentes étapes de résolution classiques.

Dans la définition suivante, nous introduisons une nouvelle notion appelée *hyper-résolution étendue*, qui étend l'hyper-résolution. L'hyper-résolution étendue peut être vue comme une application de l'hyper-résolution en plusieurs étapes.

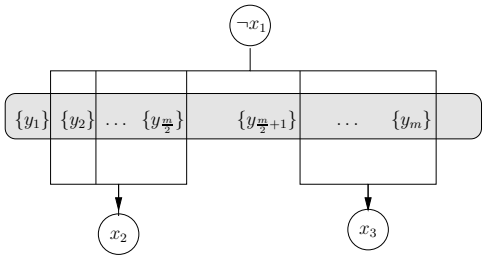
Définition 48 *Soient $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ une side clause et $c_r = (x_1 \vee x_2 \vee \cdots \vee x_k)$ une clause telles que $1 < k \leq m + 1$. Soit $c_b = c_{h_1} \wedge c_{h_2} \wedge \cdots \wedge c_{h_{k-1}}$ une formule CNF telle que $c_{h_1} = (\neg y_1 \vee x_1 \vee x_2) \wedge \cdots \wedge (\neg y_{\frac{m}{k-1}} \vee x_1 \vee x_2)$ et $c_{h_{k-1}} = (\neg y_{m-\frac{m}{k-1}+1} \vee x_1 \vee x_k) \wedge \cdots \wedge (\neg y_m \vee x_1 \vee x_k)$. Nous définissons l'hyper-résolvante étendue entre c_s et c_b comme : br(c_s, c_b) $= hr(\ldots hr(hr(c_s, c_{h_1}), c_{h_2}) \ldots c_{h_{k-1}}) = c_r$. Nous appelons $B_k^m(x_1, x_2, \ldots x_k) = c_s \wedge c_b$ une hyper-formule étendue.*

Il est important de noter que dans la définition 48, pour $k = 2$, l'hyper-résolution étendue correspond à la règle de l'hyper-résolution suivante : $hr(c_s, c_{h_1}) = (x_1 \vee x_2)$ où $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ et $c_{h_1} = (\neg y_1 \vee x_1 \vee x_2) \ldots (\neg y_m \vee x_1 \vee x_2)$.

La figure 5.2 illustre la définition 48. Un exemple d'hyper-formule étendue $B_3^m(x_1, x_2, x_3)$ et sa représentation graphique y sont donnés. Sur la partie gauche de la figure, la side clause possède un fond gris et, les autres clauses de $B_3^m(x_1, x_2, x_3)$ sont représentées en utilisant la représentation graphe-SAT,

i.e, chaque arc de x_1 à x_i avec $1 < i \leq 3$ est étiqueté avec un littéral diffèrent de la side clause. La partie droite de la figure 5.2 donne les différentes clauses faisant partie de la formule B_3^m.

Remarque 3 *En appliquant la résolution entre la side clause et les m clauses restantes, on peut dériver la nouvelle clause $c_r = (x_1 \vee x_2 \vee x_3)$ qui représente l'implication suivante $(\neg x_1 \rightarrow (x_2 \vee x_3))$. Une telle résolvante c_r peut être obtenue avec une étape de l'hyper-résolution étendue.*



$y_1 \vee y_2 \ldots \vee y_{\frac{m}{2}} \vee y_{\frac{m}{2}+1} \vee \ldots \vee y_m$
$x1 \vee \neg y_1 \vee x_2$
$x_1 \vee \neg y_2 \vee x_2$
$\ldots$
$x1 \vee \neg y_{\frac{m}{2}} \vee x_2$
$x1 \vee \neg y_{\frac{m}{2}+1} \vee x_3$
$\ldots$
$x1 \vee \neg y_m \vee x_3$

FIG. 5.2 – Une hyper-formule étendue B_3^m et l'ensemble de clauses associé

Nous souhaitons construire un graphe en connectant les hyper-formules étendues de manière hiérarchique. La formule déduite doit posséder le littéral $\neg a$ impliqué par l'hyper-résolution étendue. Le graphe associé à notre formule est schématisé dans la figure 5.3. Dans cette figure, les carrés de couleur grise représentent les side clauses. Pour raison de clarté, un littéral x_i introduit au niveau k est noté x_k^i. En commençant à partir du littéral a, nous utilisons un processus de construction incrémentale. En utilisant l'hyper-résolution étendue, à chaque étape on génère un nœud supplémentaire jusqu'à atteindre le niveau l. A ce niveau la résolvante générée obtenue par hyper-résolution étendue est de taille $l+1$. Dans la figure 5.3, la résolvante obtenue est $(\neg a \vee x_l^1 \vee x_l^2 \vee \cdots \vee x_l^l)$. À partir de ce niveau, à chaque nouvelle étape, on élimine un nœud. Le dernier est le nœud associé à $\neg a$. A ce nœud, la résolvante générée est $\neg a$. Clairement, à partir de ce graphe on peut dériver le littéral $\neg a$ par hyper-résolution étendue. Cependant, le nombre d'étapes de résolution classique est exponentiel en l. Nous formalisons cette construction dans la définition 49.

Définition 49 *Nous définissons $\mathcal{F}_l^m(a) =^C \mathcal{F}_l^m(a) \wedge^D \mathcal{F}_l^m(a)$ l'ensemble de clauses obtenu avec la représentation graphe-SAT partielle de la figure 5.3. Chaque carré gris représente une hyper-formule étendue B_3^m. $\mathcal{F}_l^m$ est appelée formule diamond.*

La propriété suivante assure que le littéral $\neg a$ est impliqué par l'ensemble de clauses $\mathcal{F}_l^m(a)$.

Propriété 15 *Soit $\mathcal{F}_l^m(a)$ une formule diamond. Nous avons $\mathcal{F}_l^m(a) \models \neg a$*

Preuve : Ébauche de la preuve : rappelons tout d'abord que pour une hyper-formule étendue $B_3^m(x_1, x_2, x_3)$ on peut dériver la clause $(x_1 \vee x_2 \vee x_3)$ (ou $\neg x_1 \rightarrow x_2 \vee x_3$) par hyper résolution étendue. En partant du littéral a (haut) vers le littéral $\neg a$ (bas) et en appliquant une hyper résolution étendue à chaque étape, on va générer les implications suivantes : $a \rightarrow x_2^1 \vee x_2^2$, $a \rightarrow x_3^1 \vee x_3^2 \vee x_3^3 \ldots$, $a \rightarrow x_{2\times l-2}^1 \vee x_{2\times l-2}^2$ et $a \rightarrow \neg a$

A partir de cette propriété, il est facile de générer des instances SAT insatisfiables en générant la formule $\mathcal{F}_l^m(a) \wedge \mathcal{F}_l^m(\neg a)$. Cependant, nous avons observé que ces instances sont en général faciles à résoudre. Afin de générer des instances difficiles, notre générateur, appelé générateur diamond, considère deux graphes différents, un pour l'implication d'un littéral a et l'autre pour l'implication d'un deuxième littéral b.

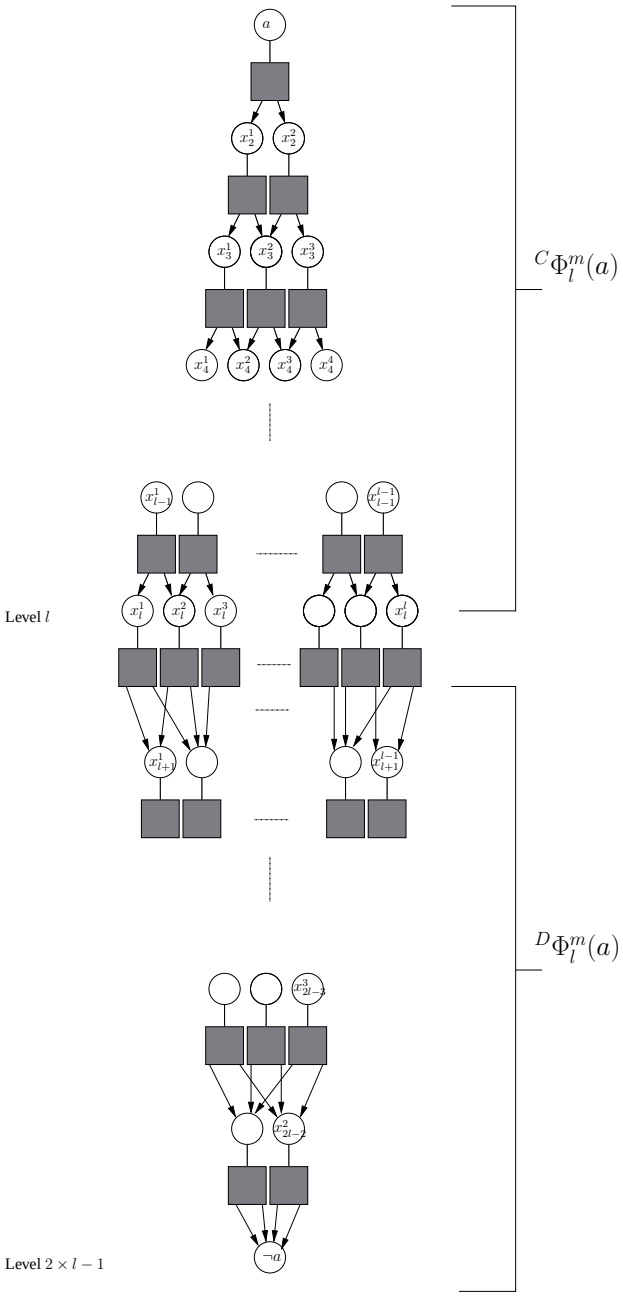


FIG. 5.3 – Formule diamond $\mathcal{F}_l^m(a)$: une hiérarchie d'hyper-formules étendues

Générateur Diamond

Les formules générées $\mathcal{G}_l^m$ sont une conjonction de deux formules diamond $\mathcal{F}_l^m(a) \wedge \mathcal{F}_l^m(b)$ où a et b sont deux littéraux différents. L'ensemble de variables de $\mathcal{G}_l^m$ est l'union de deux ensembles de variables disjoints $\mathcal{S}$ et $\mathcal{N}$. L'ensemble $\mathcal{S}$ représente l'ensemble des variables utilisées pour construire les sides clauses (carrés gris dans la figure 5.3) et, $\mathcal{N}$ représente le reste des variables (sous forme de cercles dans la figure 5.3). Les formules $\mathcal{F}_l^m(a)$ et $\mathcal{F}_l^m(b)$ sont générées en utilisant les ensembles de variables $\mathcal{S} \cup \mathcal{N} \setminus \{b\}$ et $\mathcal{S} \cup \mathcal{N} \setminus \{a\}$ respectivement.

Les formule $\mathcal{F}_l^m(a)$ et $\mathcal{F}_l^m(b)$ sont générée comme dans la figure 5.3. A chaque niveau k,

- tous les littéraux correspondant aux side clauses (carrés) et nœuds internes (cercles) sont générés de façon aléatoire à partir de l'ensemble des littéraux positifs associés à $\mathcal{S}$ et à partir de l'ensemble complet des littéraux associés à $\mathcal{N} \setminus \{a, b\}$ respectivement.
- Les deux ensembles de littéraux associés aux nœuds internes du niveau k et $k-1$ sont disjoints.

Comme on peut le voir dans la section 5.5, le générateur diamond permet de générer des instances insatisfiables de petite taille mais difficiles à résoudre. En outre, dans notre générateur seules les hyper-formules étendues de la forme B_3^m (voir définition 49) sont considérées. Par conséquent, les instances générées contiennent seulement des clauses de taille 3 et m. En utilisant d'autres types d'hyper-formules étendues, on peut générer des instances de plus grande taille. Ce générateur est disponible sur `http://www.cril.fr/~jabbour`.

5.4.3 Graphe & pré-traitement

Le pré-traitement des formules CNF est une étape de simplification très importante pour l'efficacité des solveurs SAT. Plusieurs techniques de pré-traitement ont été proposées. Citons les plus récentes comme "SatELite" qui utilise la technique de l'élimination de variables [EÉN & BIERE 2005] intégrée dans la plupart des solveurs SAT modernes et "hypre" un pré-traitement basé sur l'hyper résolution binaire [BACCHUS & WINTER 2003]. Plus récemment, un autre pré-traitement efficace a été proposé par Piette et al.[PIETTE *et al.* 2008]. Ce pré-traitement, appelé ReVivAl, produit des sous-clauses en opérant sur les redondances de la formule.

Dans cette section, nous proposons un nouveau pré-traitement basé sur une représentation graphe SAT partiel d'une formule CNF. Cette représentation considère un seul ordre pour chaque clause (ordre lexicographique). Cet ordre est utilisé pour générer un seul arc pour chaque clause.

Définition 50 (Graphe SAT partiel) *Soit $\mathcal{F}$ une formule. Nous définissons le graphe partiel $G_{\mathcal{F}}^p$ comme suit :*

- $S = \{x, \neg x | x \in \mathcal{L}(\mathcal{F})\}$
- $E = \bigcup_{c \in \mathcal{F}} arcs(c)$*, tel que* $arcs(c)$ *pour* $c = (x_1 \vee x_2 \vee .. \vee x_n)$ *est défini comme suit :*
  - $(1 \leq i < n)$ *:* $a_i = (x_i, x_{i+1}, v(a_i))$ *et* $v(a_i) = \{x_j | 1 \leq j \leq n$ *et* $j \neq i$ *et* $j \neq i+1\}$ *et*
  - $(i = n)$ *:* $a_n = (x_n, x_1, v(a_n))$ *et* $v(a_n) = \{x_2, ..., x_{n-1}\}$

Exemple 10 *Soit $\mathcal{F} = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x) \wedge (\neg x_2 \vee x) \wedge (\neg x_3 \vee x)$ une formule CNF. Le graphe SAT partiel $G_{\mathcal{F}}^p$ est représenté dans la figure 5.4. Comme on peut le remarquer, pour les clauses binaires, le graphe obtenu (graphe SAT partiel) est le même avec et sans restriction (graphe SAT), i.e, une clause binaire est représentée par deux arcs dans les deux types de graphes.*

Le graphe SAT partiel est un bon compromis entre une représentation complète (tous les arcs associés à une clause) et une représentation unique (un seul arc par clause). Comme expliqué dans les sections précédentes, l'utilisation de l'opération de transitivité sur un chemin du graphe, permet de générer facilement des résolvantes. L'application de la saturation par la résolution classique revient à appliquer la

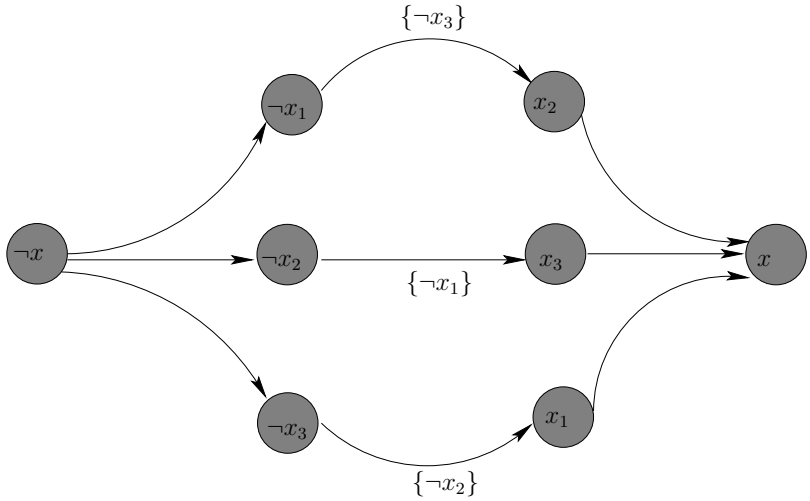


FIG. 5.4 – Graphe SAT partiel $G^p_{\mathcal{F}}$ associé à la formule $\mathcal{F}$ (exemple 10)

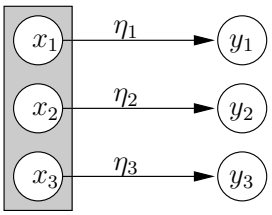
fermeture transitive du graphe. En conséquence, un tel processus est clairement exponentiel dans le pire des cas.

Décrivons maintenant l'étape principale de notre pré-traitement utilisant le graphe partiel représentatif d'une formule CNF $\mathcal{F}$. Le but de cette étape est de générer une résolvante fondamentale de taille limitée. A chaque itération une nouvelle clause $c = (x_1 \vee x_2 \vee \cdots \vee x_n)$ de $\mathcal{F}$ est sélectionnée et traitée comme décrit dans l'algorithme 8. Pour chaque $x_i \in c$ (ligne 4), un arc $a = (x_i, x_j, v(a)) \in G^p_{\mathcal{F}}$ est choisi tel que $res(c, (\neg x_i \vee \neg v(a) \vee x_j), x_i)$ est fondamentale (ligne 4 à 11). Plus précisément, la nouvelle résolvante est générée suivant les arcs de $G^p_{\mathcal{F}}$. Une telle résolvante est composée des ensembles de littéraux cumulés ($nuds$) et contextes (η). Un arc a est choisi tel que $res(c, cla(a), x_i)$ est fondamentale (ligne 5 et 6). Si un tel arc existe, le contexte η (resp. $nuds$) est augmenté de $a(v)$ (resp. $\{x_j\}$) (ligne 7) sinon x_i est simplement ajouté à l'ensemble des nœuds (ligne 9) si la résolvante obtenue est fondamentale (test ligne 8). Dans le dernier cas, si aucune clause fondamentale n'a pu être générée, le pré-traitement s'arrête (ligne 11).

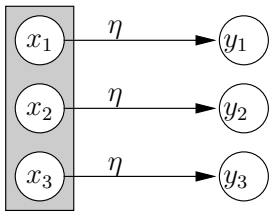
Avant de donner les différentes formes d'hyper résolution, nous définissons ci-dessous la forme la plus générale d'hyper résolution.

Définition 51 (Hyper-résolution générale) *Soient $c_s = (x_1 \vee x_2 \vee \cdots \vee x_n)$ une side clause et, $c_b = \{(\neg x_1 \vee \alpha_1), \ldots, (\neg x_n \vee \alpha_n)$ un ensemble de clauses, où α_i pour $1 \leq i \leq n$ sont des sous-clauses. La résolvante $\cup_{1 \leq i \leq n} \alpha_i$ est appelée une hyper résolvante générale de c_s et c_b (notée $hrg(c_s, c_b)$).*

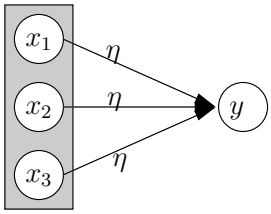
La figure 5.5 nous donne une représentation graphique des différentes formes de l'hyper résolution. Pour une formule contenant une clause $c = (x_1 \vee x_2 \vee x_3)$ et un ensemble de clauses binaires $C = \{(\neg x_1 \vee y), (\neg x_2 \vee y), (\neg x_3 \vee y)\}$, le littéral y est généré par l'algorithme 7. A partir de la clause c et des arcs qui codent C. Dans ce cas, l'application de notre algorithme correspond exactement à l'application de l'hyper résolution binaire sur c et C [BACCHUS & WINTER 2003] (voir figure 5.5.d). Pour l'ensemble de clauses $C = \{(\neg x_1 \vee \neg \eta \vee y), (\neg x_2 \vee \neg \eta \vee y), (\neg x_3 \vee \neg \eta \vee y)\}$) (voir figure 5.5.c), notre algorithme génère par hyper résolution la clause $(\neg \eta \vee y)$. Pour l'ensemble de clauses $C = \{(\neg x_1 \vee \neg \eta \vee y_1), (\neg x_2 \vee \neg \eta \vee y_2), (\neg x_3 \vee \neg \eta \vee y_3)\}$ (voir figure 5.5.b), notre algorithme génère par hyper résolution étendue la clause $(\neg \eta \vee y_1 \vee y_2 \vee y_3)$. Dans la figure 5.5.b, nous présentons l'hyper résolution générale entre c et $C = \{(\neg x_1 \vee \neg \eta_1 \vee y_1), (\neg x_2 \vee \neg \eta_2 \vee y_2), (\neg x_3 \vee \neg \eta_3 \vee y_3)\}$. La résolvante obtenue est $(\neg \eta_1 \vee \neg \eta_2 \vee \neg \eta_3 \vee y_1 \vee y_2 \vee y_3)$.



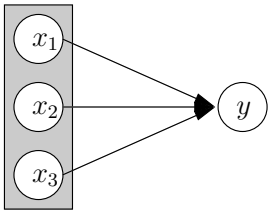
a. Hyper-résolution générale



b. Hyper-résolution étendue



c. Hyper-résolution



d. Hyper-résolution binaire

FIG. 5.5 – Hyper-Res générale /Hyper-Res étendue/ Hyper-Res / Hyper-Res Bin : une comparaison

Pour une formule $\mathcal{F}$, notre pré-traitement (algorithme 8) consiste à itérer l'algorithme 7 pour chaque clause $c \in \mathcal{F}$. Le processus est réitéré sur la résolvante générée (par l'appel à hypResGen Algorithme 7) tant qu'elle est fondamentale et que sa taille ne dépasse pas une valeur maximale (maxRes) fixée à l'avance.

5.5 Expérimentations

Toutes les expérimentations menées dans cette section ont été réalisées sur un Xeon 3.2 GHz (2 GB RAM). Ces expérimentations ont été menées afin de valider les différentes utilisations faites à partir de notre représentation sous forme de graphe. Nous commençons par évaluer la taille de l'ensemble 2-SAT strong backdoors généré (section 5.4.1) sur diverses instances des dernières compétitions SAT. Ensuite, nous testons le générateur d'instances difficiles (section 5.4.2) en utilisant divers solveurs *état de l'art*. Enfin, notre pré-traitement (section 5.4.3) est utilisé sur des instances des dernières compétitions SAT et également sur les instances SAT générées par notre générateur Diamond.

5.5.1 Calcul d'ensembles 2-SAT Strong Backdoors

La table 5.1 présente une comparaison entre notre approche pour calculer l'ensemble 2-SAT strong backdoors (algorithme 6) et celle calculant l'ensemble Horn strong backdoors proposée récemment dans [PARIS *et al.* 2006]. Pour chaque instance, le nombre de variables (#V), le nombre de clauses (#C) et la taille de l'ensemble 2-SAT (resp. Horn) strong backdoors sont indiqués. Le temps nécessaire au calcul de l'ensemble strong backdoors n'est pas indiqué car il n'excède jamais 10 secondes. Les résultats montrent clairement que notre algorithme calcule des ensembles strong backdoors de plus petite taille que celui proposé dans [PARIS *et al.* 2006]. Dans certains cas, le gain est très important (voir les séries `mm-*`, `equilarge-*`, `iso-*`, `series-*`). Par ailleurs, dans de nombreux cas, les ensembles strong backdoors générés sont de très petite taille (series `equilarge` et `iso`). Bien entendu, les plus mauvais résultats sont obtenus sur les instances aléatoires (série `mod2c`). Pour conclure, notre représentation graphe-SAT fournit une façon très intuitive mais efficace pour calculer des ensembles strong backdoors.

Algorithm 7: HypResGen : une étape du pré-traitement

Input: $G^p(V, E)$ la représentation en graphe de $\mathcal{F}$
$c \in \mathcal{F}$
Output: c_{res} une clause résolvante

begin
  $\eta = \emptyset$
  $nuds = \emptyset$
  foreach $x_i \in c$ **do**
    if $\exists a = (x_i, x_j, v(a)) \in E$ *tel que* $x_j \notin \eta$ *et* $v(a) \cap nuds = \emptyset$ *et* $\neg x_j \notin c$ *et* $\neg v(a) \cap c = \emptyset$
    then
    return $\eta = \eta \cup v(a)$, $nuds = nuds \cup \{x_j\}$
    else if *(*$x_i \notin \eta$ *et* $\neg x_i \notin nuds$*)*
    then $nuds = nuds \cup \{x_i\}$
    else
    return *null*
    else
    return *null*
    $c_{res} = nuds \cup \neg\eta$
  end
  return c_{res}
end

Algorithm 8: HyReGenPre : un pré-traitement par hyper-résolution générale

Input: $\mathcal{F}$ la formule CNF et G^p le graphe SAT partiel associé ;
Output: $\mathcal{F}'$ la formule pré-traitée;

begin
  $\mathcal{F}' \leftarrow \mathcal{F}$;
  foreach $c \in \mathcal{F}$ **do**
    $fin \leftarrow faux$
    while *(* $(\neg fin)$ *et (*$|c| < maxRes$*)* **do**
      $c \leftarrow hyResGen(G^p, c)$;
      if *(*$c == null$*)* **then return** $fin \leftarrow vrai$
      else $\mathcal{F}' \leftarrow \mathcal{F}' \cup c$;
    end
    return $\mathcal{F}'$;
  end
end

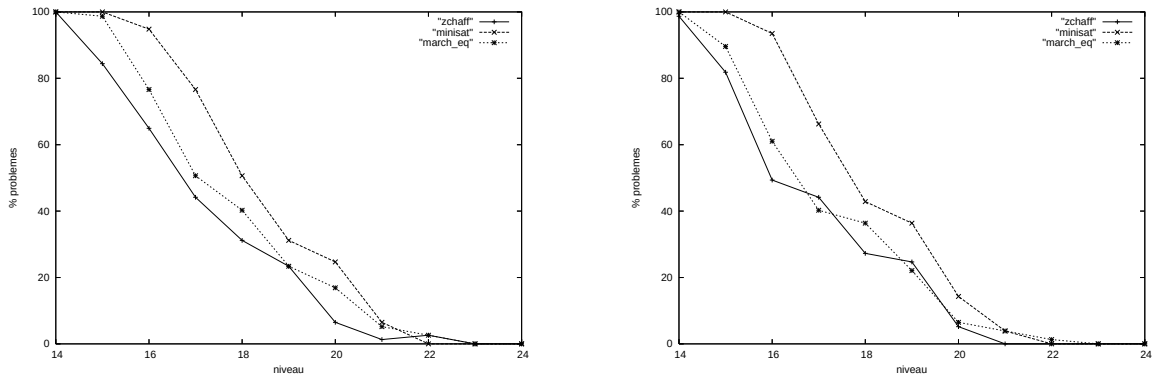
instance	#V	#C	Horn BD	2SAT BD
clauses-4-1967	267767	823658	–	60747 (22%)
clauses-2-1966	75528	226124	29357 (38%)	17945 (23%)
depots1_ks99i-4010	161	445	32 (19%)	31 (19%)
equilarge_l2-519	4487	18555	2582 (57%)	367 (8%)
equilarge_l3-520	4496	18591	2595 (57%)	373 (8%)
equilarge_l1-518	4574	18903	2635 (57%)	374 (8%)
ferry6_ks99i.renamed-sat05-3997	1425	14454	657 (46%)	539 (35%)
ferry8_ks99a.renamed-sat05-4004	1259	15206	573 (45%)	471 (37%)
linvrinv7-566	1274	4166	631 (49%)	467 (36%)
linvrinv8-567	1920	6337	961 (50%)	705 (36%)
linvrinv5-564	450	1426	221 (49%)	167 (37%)
iso-brn100-3025	3587	5279	372 (10 %)	349 (9%)
iso-brn009-2934	1770	4540	318 (17 %)	187 (10%)
iso-icl009-3243	2251	5177	442 (19 %)	512 (22%)
iso-ukn006-3387	1906	4188	407 (21%)	453 (23%)
iso-icl008-3242	2136	5938	461 (21%)	545 (25%)
mm-2x3-8-8-s.1-476	1148	8088	621 (54%)	132 (11%)
mm-2x3-8-8-sb.1-475	2326	80891	1661 (71%)	926 (39%)
mod2-rand3bip-unsat-120-1-2624	120	320	85 (69%)	50 (41%)
mod2c-rand3bip-unsat-120-3-2340	160	640	126 (78%)	86 (53%)
mod2c-rand3bip-unsat-150-1-2368	200	800	154 (77%)	111 (55%)
mod2c-rand3bip-unsat-105-2-2324	140	560	109 (77%)	78 (55%)
mod2c-rand3bip-unsat-120-1-2338	160	640	126 (78%)	90 (56%)
mod2-rand3bip-unsat-150-2-2655	150	400	106 (70%)	61 (40%)
par32-3-c	1325	5294	698 (52%)	199 (15%)
par32-1-c	1315	5254	696 (52%)	202 (15%)
par32-4-c	1333	5326	703 (52%)	202 (15%)
par32-5-c	1339	5350	708 (52%)	202 (15%)
par16-5-c	341	1360	189 (55%)	70 (20%)
par16-1-c	317	1264	176 (55%)	68 (21%)
pmg-12-UNSAT-3940	190	632	136 (71%)	77 (40%)
pmg-14-UNSAT-3942	577	1922	408 (70%)	232 (40%)
pmg-11-UNSAT-3939	169	562	122 (72%)	65 (38%)
pmg-13-UNSAT-3941	409	1362	291 (71%)	163 (39%)
series18	6410	32421	1665 (25%)	1122 (17%)
series16	4546	22546	1189 (26%)	870 (19%)
strips-gripper-08t14-1156	1966	23326	826 (42%)	773 (39%)
strips-gripper-16t31-1146	8904	222172	4176 (46%)	3788 (42%)
strips-gripper-18t35-1147	11318	316416	5331 (47%)	4835 (42%)
strips-gripper-10t19-1143	3390	53008	1475 (43%)	1400 (41%)
satellite3_v01a-3989	303	7840	226 (74%)	210 (69%)

TAB. 5.1 – 2-SAT vs Horn strong backdoors

De plus, ces résultats montrent que l'utilisation du fragment polynomial 2-SAT permet de déterminer des ensembles strong backdoors plus intéressants que le fragment Horn-SAT.

5.5.2 Évaluation du Générateur Diamond

Notre générateur permet d'obtenir différentes classes d'instances $\mathcal{G}_l^m$ suivant les valeurs utilisées pour m (taille de la side clause) et pour l (niveau). Dans nos expérimentations, deux classes d'instances $\mathcal{G}_l^4$ et $\mathcal{G}_l^5$ ont été considérées. Pour chacune d'entre elles, le nombre de niveaux (l) varie entre 14 et 24. Pour un m fixé et l différentes formules ont été générées en variant le nombre de variables. Pour un nombre de variables, niveau et taille de la side clause donnés, 100 instances ont été générées. Afin de vérifier la difficulté des instances générées, trois solveurs *état de l'art* ont été considérés. Les deux premiers sont deux solveurs de types CDCL (Conflict Driven Clause Learning) : ZCHAFF (versionMARCH_EQ 2007.3.12) et MINISAT (version 2.0). Le dernier est MARCH_EQ (version 010), un solveur de type look-ahead. Une comparaison de ces trois solveurs sur les instances proposées est effectuée.

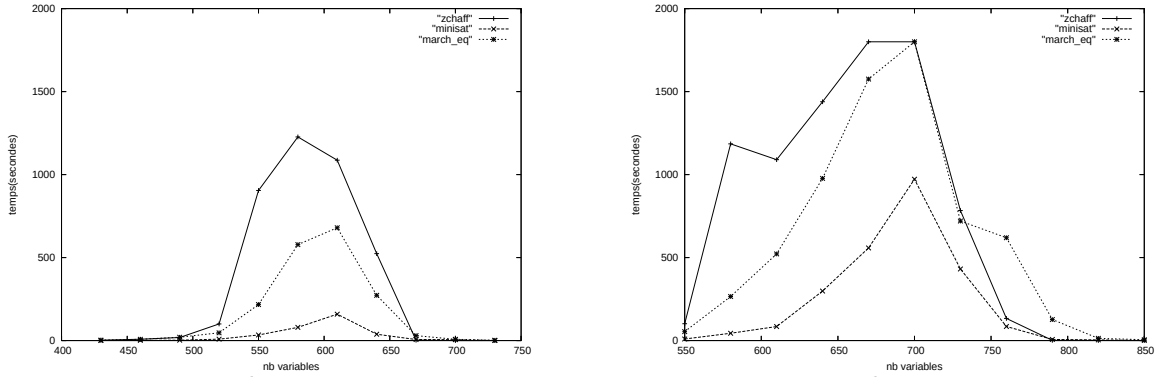


a. $m = 4$ b. $m = 5$

FIG. 5.6 – Pourcentage d'instances résolues par rapport au niveau l

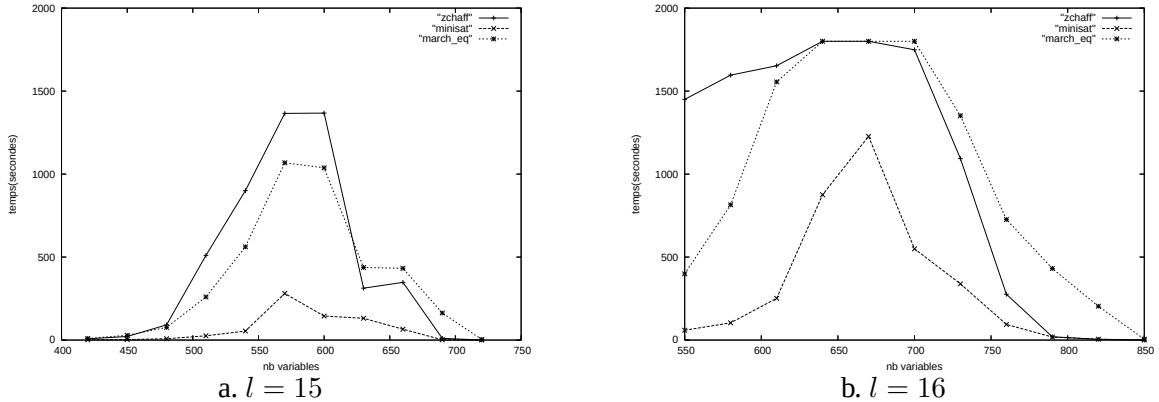
La figure 5.6 donne un pourcentage des instances résolues par chacun des trois solveurs pour différents niveaux. La figure 5.6.a montre les résultats obtenus pour $m = 4$. Il est clair que la difficulté des instances générées est proportionnelle à la valeur du niveau l. A partir du niveau $l = 16$, les instances générées deviennent très difficiles pour les trois solveurs. Au niveau $l = 23$ aucun des solveurs considérés n'arrive à résoudre une instance. Notons que, à ce niveau, les instance générées contiennent moins de 6000 clauses et que le nombre de variables n'excède pas 2000. Pour $m = 5$ (figure 5.6.b), les instances générées sont plus difficiles qu'au niveau $m = 4$. En effet, au niveau $l = 20$, ZCHAFF et MARCH_EQ résolvent 5% des instances. alors que MINISAT résout 15%. Ici le nombre de variables est inférieur à 1700.

Pour un niveau donné, on voulait connaître l'impact du nombre de variables sur la difficulté des instances générées. Comme montré sur la figure 5.7 (où $m = 4$), on observe un phénomène de seuil quand le nombre de variables augmente. En effet, pour un niveau donné, il existe une valeur critique du nombre de variables où l'instance considérée avec cette valeur est la plus difficile. Ce nombre est proche de 600 pour le niveau $l = 15$ (figure 5.7.a) et proche de 700 pour le niveau $l = 16$. Nous pouvons aussi observer que MINISAT est le meilleur solveur sur ces instances et ZCHAFF est celui qui obtient les plus mauvais résultats. Ce phénomène de seuil est observé aussi pour les instances $\mathcal{G}_l^5$ comme on peut le voir sur la figure 5.8. Dans ce cas, le point critique est proche de 580 variables pour le niveau $l = 15$ alors qu'il est proche de 680 pour le niveau $l = 16$.



a. $l = 15$ b. $l = 16$

FIG. 5.7 – Temps de résolution ($m = 4$)



a. $l = 15$ b. $l = 16$

FIG. 5.8 – Temps de résolution($m = 5$)

Les tables 5.2 et 5.3 montrent que notre générateur permet d'obtenir des petites instances SAT et UNSAT. Le temps moyen indiqué dans ces figures tient seulement compte du temps obtenu lorsque les instances sont résolues. Pour un niveau l donné, on arrive à produire des instances satisfaisables intéressantes et qui sont difficiles pour les trois solveurs (table 5.2), mais aussi des instances insatisfiables difficiles comme l'exhibe la table 5.3.

		ZCHAFF		MINISAT		MARCH_EQ	
niveau l	nb inst.	nb résolues	temps moyen	nb résolus	temps moyen	nb résolues	temps moyen
15	63	58	68	**63**	37	60	118
16	78	65	92	**78**	113	62	218
17	84	67	96	**84**	139	67	169
18	68	45	137	**68**	199	59	327
19	52	37	240	**52**	241	35	247
20	30	1	326	**30**	421	18	359
21	8	1	872	**8**	440	7	289
22	3	-	-	-	-	**3**	1035

TAB. 5.2 – Petites instances SAT difficiles ($m = 4, 5$)

		ZCHAFF		MINISAT		MARCH_EQ	
niveau l	nb inst.	nb résolues	temps moyen	nb résolues	time moyen	nb résolues	temps moyen
14	106	106	67	**106**	5	106	40
15	91	70	155	**91**	55	85	214
16	67	23	592	**67**	254	44	523
17	26	0	-	**26**	834	3	1203
18	4	0	-	**4**	1383	-	-

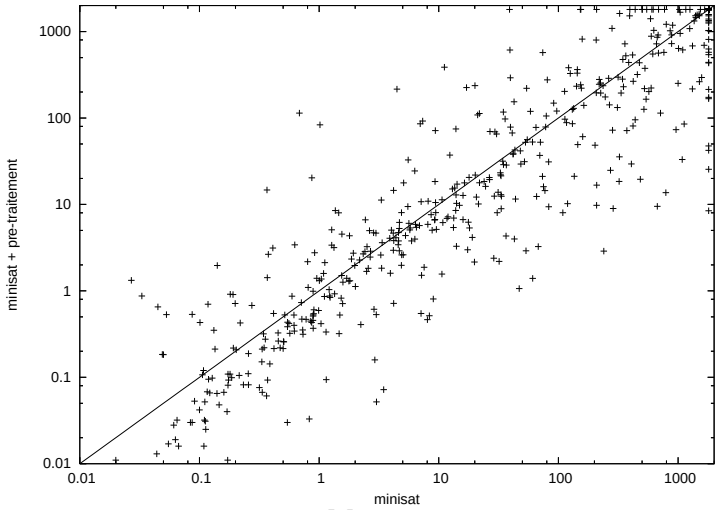
TAB. 5.3 – Petites instances UNSAT difficiles ($m = 4, 5$)

Les expérimentations menées montrent qu'on obtient des instances intéressantes à étudier et qui constituent un vrai challenge pour les solveurs SAT existants.

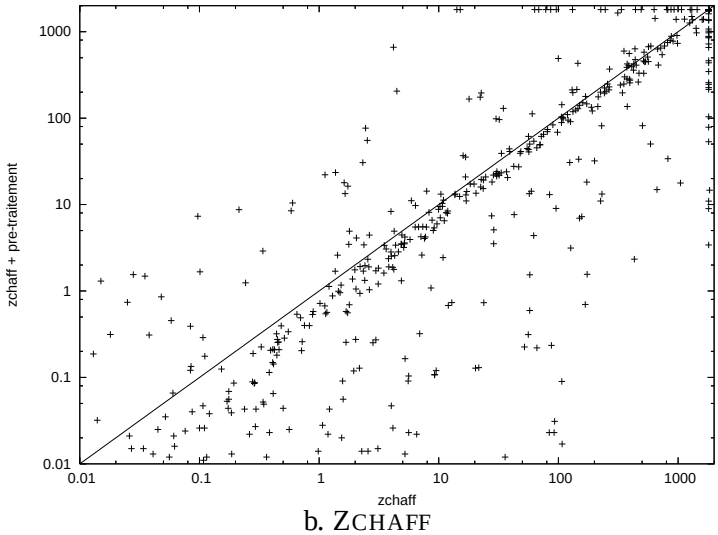
5.5.3 Évaluation du pré-traitement

Dans cette section, une évaluation expérimentale du pré-traitement proposé est montrée sur les instances générées en utilisant le générateur Diamond et sur des instances des dernières compétitions SAT. Nous commençons par évaluer l'impact de ce pré-traitement sur 2000 instances générées par le générateur Diamond en faisant varier la taille de la side clause (m), le niveau (l) et le nombre de variables (voir section 5.5.2).

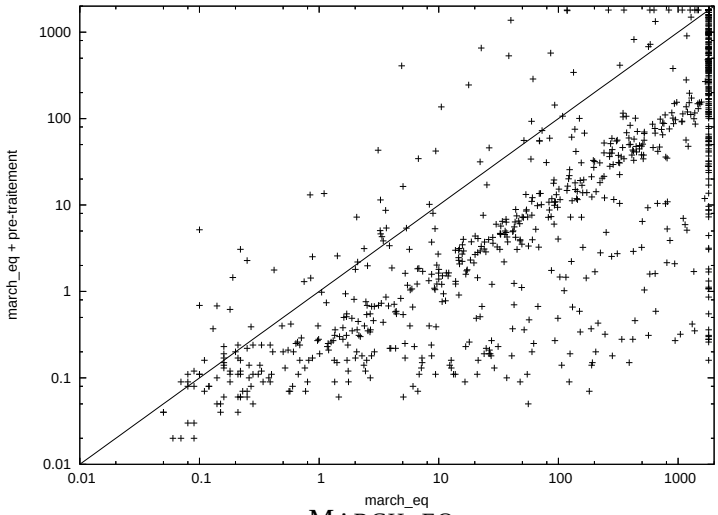
La figure 5.9 compare les résultats des trois solveurs testés avec et sans ce pré-traitement. Chaque scatter plot donné dans la figure 5.9 illustre les résultats comparatifs des trois solveurs sur chaque instance. L'axe des abscisses (resp. des ordonnées) correspond au temps de calcul tx (resp. ty) obtenu par chaque solveur sur l'instance originale (resp. instance obtenue après le pré-traitement). Chaque coordonnée (tx, ty) correspond donc à une instance SAT. Les points au-dessus (respectivement au-dessous) de la diagonale indiquent que la formule originale est résolue plus rapidement, i.e, $tx < ty$ (respectivement moins rapidement, i.e, $tx > ty$) que la formule simplifiée. Ce processus de pré-traitement améliore significativement les performances des trois solveurs (MARCH_EQ, ZCHAFF and MINISAT). Notons que les instances où les performances diminuent correspondent majoritairement aux instances satisfaisables.



a. MINISAT



b. ZCHAFF



c. MARCH_EQ

FIG. 5.9 – Comparaison des temps de calcul avec et sans pré-traitement sur les formules diamond

Finalement, le plot de la figure 5.10 est obtenu comme suit : l'axe des abscisses représente le nombre d'instances résolues par un solveur donné et l'axe des ordonnées le temps nécessaire pour résoudre ce nombre d'instances. Dans la figure on tient compte de l'ensemble des instances générées ($\mathcal{G}_l^4$ et $\mathcal{G}_l^5$). Il est clair que ZCHAFF et MARCH_EQ ne sont pas très efficaces sur ce type d'instances. MINISAT semble être le plus performant parmi les trois solveurs. En ajoutant le pré-traitement, les performances changent de façon spectaculaire. Chaque solveur est capable de résoudre plus d'instances avec les nouvelles clauses ajoutées à la base et obtenues par pré-traitement (e.g. jusqu'à 180 instances pour MARCH_EQ). En outre, dans ce cas, MARCH_EQ devient le meilleur solveur.

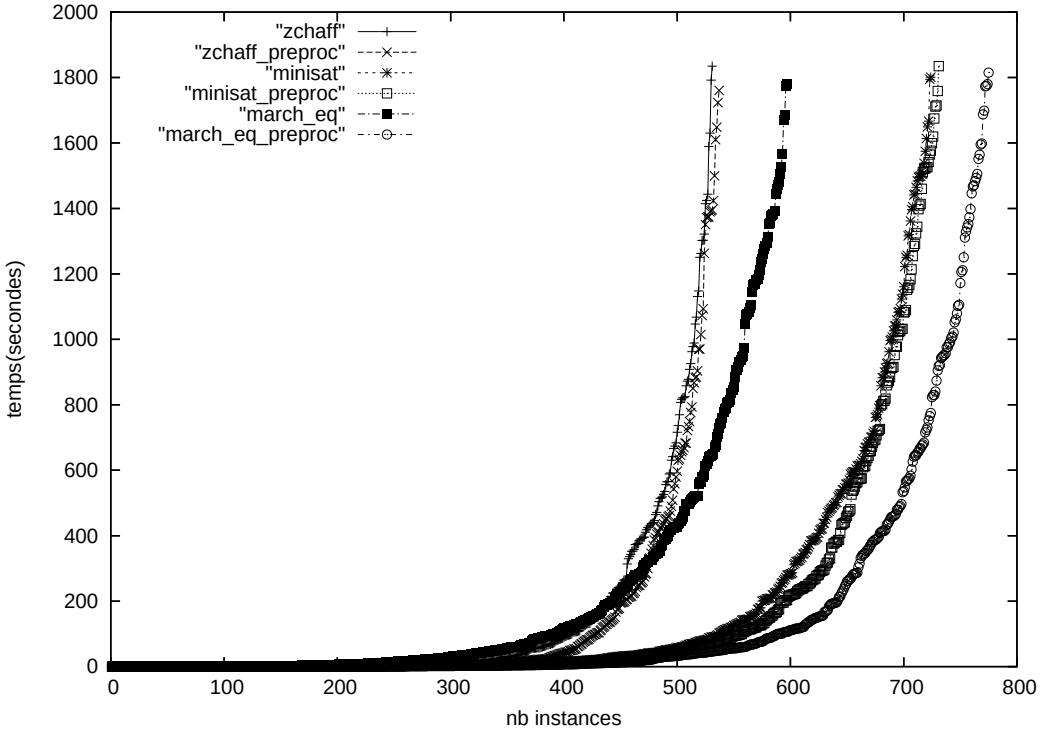


FIG. 5.10 – Nombre d'instances résolues par rapport au temps CPU

Nous avons également expérimenté notre pré-traitement sur des instances SAT des dernières compétitions. La taille des instances étant très grande, nous avons limité le pré-traitement à 10% des clauses de l'instance et nous avons fixé la taille maximum des résolvantes ($maxRes$) à 100.

La table 5.4 montre une comparaison expérimentale du solveur Minisat [EÉN & SÖRENSSON 2003] avec et sans notre pré-traitement. Nous comparons également ce pré-traitement avec SatELite, considérée comme l'une des meilleures techniques de pré-traitement. Le temps de calcul est donné en secondes et limité à 900 secondes. Pour chaque instance, nous indiquons le nombre de variables (#V), le nombre de clauses (#C), le nombre de clauses ajoutées à la formule originale (#CA), le temps dû au pré-traitement (pré-traitement) et le temps total (pré-traitement + résolution) mis par Minisat et par MINISAT +SatELite. Ici aussi, les résultats sont prometteurs et montrent clairement l'intérêt de faire ce genre de pré-traitement. Par exemple, notre pré-traitement permet de résoudre 12 instances de plus que Minisat classique. L'utilité de telles résolvantes est clairement mis en évidence (e.g. `strips-gripper-*`, `homer-*` et `series-*`). Bien évidemment, concernant les instances faciles, notre pré-traitement décroît les performances de Minisat. De plus, notre pré-traitement soutient la comparaison avec SatELite que ce soit sur des instances aléatoires (`mod`) ou structurées (`vmpc`).

instance	SAT	#V	#C	#CA	preproc	Minisat +preproc	Minisat	Minisat + SatELite
mod2-sat-280-1.as.sat05-2263	Y	280	1120	183	0.76	**574**	–	–
mod2-sat-240-1.as.sat05-2203	Y	240	960	162	0.73	**250**	–	–
mod2-sat-270-1.as.sat05-2248	Y	270	1080	180	0.76	**510**	–	–
mod2-sat-260-2.as.sat05-2234	Y	260	1040	190	0.74	**657**	–	–
mod2c-sat-210-2.as.sat05-2474	Y	297	2092	341	1.14	488	–	**319**
mod2c-sat-180-2.05-2429	Y	252	1784	303	1.08	204	411	**7**
mod2c-sat-170-3.05-2415	Y	240	1724	276	1.04	286	368	**64**
mod2-unsat-150-1.05-2654	N	150	400	79	0.52	**160**	189	784
mod2-unsat-135-2.05-2640	N	135	360	71	0.50	**336**	828.83	–
mod2-unsat-135-1.05-2639	N	135	360	70	0.51	**529**	603	–
mod2-sat-250-1.05-2218	Y	250	1000	169	0.72	**699**	753	849
fclqcolor-08-05-06.as.sat05-1273	N	116	1362	96	0.77	**36**	66	86
fphp-012-011.05-1228	N	132	1398	15	0.78	889	**651**	–
gensys-icl009.as.sat05-3135	N	926	17577	1375	6.63	759	–	**584**
gensys-icl008.as.sat05-3134	N	960	17640	1482	6.74	889	–	**763**
gensys-icl006.05-2718	N	1321	7617	2360	70.78	189	2032	**108**
gensys-ukn001.05-2678	N	1886	7761	2260	70.57	655	728	**81**
gensys-ukn006.05-3349	N	874	16339	1203	5.56	410	365	**242**
gt-020.as.sat05-1305	N	400	7050	770	2.33	**821**	–	–
gt-018.as.sat05-1292	N	324	5066	577	1.81	**18**	127	128
grid-pbl-0060.05-1342.05-1342	N	3660	7142	1483	3.98	350	192	**1**
homer11.shuffled	N	220	1122	28	0.71	**48**	59	70
homer09.shuffled	N	270	1920	1920	5.18	57	70	**52**
lisa21_0_a	Y	1453	7967	1401	2.92	32	77	**1**
php-012-012.as.sat05-1158	Y	144	804	23	0.63	66	210	**1**
strips-gripper-14t27.as.sat05-1145	Y	6778	93221	53	48.00	**199**	–	–
strips-gripper-12t23.as.sat05-1144	Y	4940	148817	38	26	167	–	**69**
vmpc_34.as.sat05-1958	Y	1156	194072	21109	53.00	**60**	–	–
vmpc_26.as.sat05-1946	Y	676	86424	9612	22.00	**209**	306	–
vmpc_27.as.sat05-1947	Y	729	96 849	10729	24.00	596	254	**182**
clqcolor-16-11-11.05-1245	Y	472	15427	1300	2.15	4.17	64	**1**
clqcolor-08-05-0605-1249	N	116	1114	94	4.8	48	85	**45**
series16	Y	4546	22546	4298	10.43	**51**	316	142
clus-1200-4800-003.03-1082	Y	1200	4800	1004	5.82	**85**	197	303
clus-1200-4800-003.as.sat03-1087	Y	1208	4800	982	6.17	**77**	174	221

TAB. 5.4 – Comparaison entre MINISAT, MINISAT + pré-traitement et MINISAT + SatELite

5.6 Conclusion

Dans ce chapitre nous avons présenté une nouvelle représentation de formules CNF sous forme de graphe. Elle étend le graphe d'implication 2-SAT. Cette nouvelle représentation offre des perspectives intéressantes. La structure de la formule (dépendances entre les variables) est clairement mieux exprimée. Nous avons montré que la résolution peut être appliquée à ce graphe en utilisant la notion de fermeture transitive du graphe. Nous avons étudié trois façons d'exploiter ce graphe. Une technique d'extraction d'ensembles 2-SAT strong backdoors est proposée, améliorant des résultats précédemment obtenus.

Un générateur particulier d'instances SAT a été proposé, s'inspirant de cette représentation sous forme de graphe. Nous avons mis en évidence sur ces instances une relation expérimentale entre la distribution des variables et la niveau du graphe.

Un pré-traitement des formules booléennes sous forme CNF est proposé étendant l'hyper-binaire résolution. L'intégration de ce pré-traitement montre une amélioration des résultats du solveur MINISAT sur une large collection d'instances.

Finalement, cette nouvelle représentation sous forme de graphe offre des perspectives intéressantes pour les recherches futures. Parmi elles, nous envisageons de calculer les conditions minimales sous lesquelles un littéral est impliqué (le plus court chemin entre un littéral et son opposé). Nous comptons également exploiter ce graphe pour extraire des informations sur la façon de faire de la résolution.

Deuxième partie

QBF

6

Formules Booléennes Quantifiées

Sommaire

6.1 Syntaxique 76
6.2 Sémantique 79
6.3 Règles de simplification 80
6.4 Algorithmes de résolution pour QBF 83
6.4.1 DPLL pour QBF 83
6.4.2 Apprentissage 84
6.5 Transformations : de QBF vers SAT 87
6.5.1 Élimination de variables dans les QBF 88
6.5.2 Résolution et extension 88
6.5.3 Élimination symbolique des quantificateurs 89
6.5.4 QBF et fonctions de skolem 90
6.5.5 QBF et codage en BDD 92

Dans ce chapitre, nous introduisons les formules booléennes quantifiées (QBFs). Après une description formelle de la syntaxe et de la sémantique des QBFs, nous présentons un panorama complet des techniques de résolution associées. Ces techniques peuvent être partitionnées en deux catégories : les approches énumératives et les symboliques. Pour la première catégorie, nous décrivons les solveurs basés sur l'approche $\mathcal{Q} - \mathcal{DPLL}$ qui est une adaptation de l'algorithme DPLL pour SAT. L'adaptation de l'apprentissage est discuté en détail, plus particulièrement l'apprentissage à partir des solutions qui est un cas spécifique aux QBFs. Pour les approches symboliques, nous donnons l'inventaire de ces techniques qui sont généralement basées sur la transformation d'une QBF vers une simple formule propositionnelle. On distingue deux approches principales : l'élimination de variables par Q-résolution comme pour le cas SAT et l'utilisation des fonctions de skolem.

6.1 Syntaxique

Définition 52 ($QPROP_{\mathcal{P}}$) *Soit $\mathcal{P}$ le langage de la logique propositionnelle classique.*
L'ensemble $QPROP_{\mathcal{P}}$ de formules booléennes quantifiées est défini inductivement à partir de l'ensemble des symboles de $\mathcal{P}$ auquel on ajoute les quantificateurs $\forall$ et $\exists$ comme suit :

1. *les constantes booléennes $vrai$ (ou $\top$) et $faux$ (ou $\bot$) appartiennent à $QPROP_{\mathcal{P}}$*
2. *chaque variable de $\mathcal{P}$ appartient à $QPROP_{\mathcal{P}}$.*

3. *si $\mathcal{F}$ et $\mathcal{F}'$ appartiennent à $QPROP_{\mathcal{P}}$, alors $(\neg\mathcal{F}), (\mathcal{F} \wedge \mathcal{F}'), (\mathcal{F} \vee \mathcal{F}'), (\mathcal{F} \rightarrow \mathcal{F}'), (\mathcal{F} \leftrightarrow \mathcal{F}')$ appartiennent à $QPROP_{\mathcal{P}}$.*
4. *si $\mathcal{F}$ appartient à $QPROP_{\mathcal{P}}$ et x appartient à $\mathcal{P}$, alors $\forall x(\mathcal{F})$ et $\exists x(\mathcal{F})$ appartiennent à $QPROP_{\mathcal{P}}$.*

Chaque formule booléenne quantifiée est obtenue en appliquant un nombre fini de fois les règles précédentes.

Exemple 11 *La formule suivante est une QBF : $\mathcal{F} = \exists y_1(((\forall x_1(\neg x_1 \vee y_1)) \vee ((\neg x_1) \vee y_1)) \wedge (y_1 \vee ((\neg x_1) \wedge z_1)))$*

Proposition 1 *Soient $\mathcal{F}$, $\mathcal{F}'$ des formules de $QPROP_{\mathcal{P}}$, x et y des variables de $\mathcal{P}$. On a les propriétés suivantes :*

1. *Si $x \notin \mathcal{V}(\mathcal{F})$, alors $\forall x\mathcal{F} \equiv \exists x\mathcal{F} \equiv \mathcal{F}$.*
2. *$\forall x(\forall y\mathcal{F}) \equiv \forall y(\forall x\mathcal{F})$*
3. *$\exists x(\exists y\mathcal{F}) \equiv \exists y(\exists x\mathcal{F})$*
4. *$\forall x(\mathcal{F} \wedge \mathcal{F}') \equiv (\forall x\mathcal{F}) \wedge (\forall x\mathcal{F}')$*
5. *$\exists x(\mathcal{F} \vee \mathcal{F}) \equiv (\exists x\mathcal{F}) \vee (\exists x\mathcal{F}')$*
6. *$\exists x\mathcal{F} \equiv \neg(\forall x(\neg\mathcal{F}))$*
7. *Si $x \notin \mathcal{V}(\mathcal{F})$, alors $\forall x(\mathcal{F} \vee \mathcal{F}') \equiv F \vee (\forall x\mathcal{F}')$*
8. *Si $x \notin \mathcal{V}(\mathcal{F})$, alors $\exists x(\mathcal{F} \wedge \mathcal{F}') \equiv \mathcal{F} \wedge (\exists x\mathcal{F}')$*

En général, nous n'avons ni $\forall x(\mathcal{F} \vee \mathcal{F}') \equiv (\forall x\mathcal{F}) \vee (\forall x\mathcal{F}')$, ni $\exists x(\mathcal{F} \wedge \mathcal{F}') \equiv (\exists x\mathcal{F}) \wedge (\exists x\mathcal{F}')$ (contre-exemple : $\mathcal{F} = x$ et $\mathcal{F}' = \neg x$).
S'il est possible d'inverser deux quantificateurs identiques et successifs, à l'inverse, il n'est pas possible d'inverser deux quantifications successives de nature différente en préservant la sémantique. Ainsi, $\forall x(\exists y\mathcal{F})$ est une conséquence logique de $\exists y(\forall x\mathcal{F})$ mais les deux formules ne sont pas équivalentes (contre-exemple $\mathcal{F} = \forall x\exists y(\neg x \vee y) \wedge (x \vee \neg y)$).

Les occurrences de variables propositionnelles x d'une QBF $\mathcal{F}$ peuvent être partitionnées en trois ensembles : les occurrences quantifiées, liées et libres de x. Les occurrences *quantifiées* sont celles apparaissant dans une quantification, i.e, juste après un quantificateur $\forall$ ou $\exists$. Dans toute sous-formule de $\forall x\mathcal{F}$ (resp. $\exists x\mathcal{F}$), toutes les occurrences de x dans $\mathcal{F}$ sont *liées* ; de telles occurrences de x sont dites *dans la portée de la quantification* de $\forall x$ (resp. $\exists x$). Enfin, les occurrences restantes dans $\mathcal{F}$ sont *libres*. Une variable $x \in \mathcal{P}$ est $libre$ dans $\mathcal{F}$ si et seulement si x a une occurrence libre dans $\mathcal{F}$.

Exemple 12 *Considérons la* QBF *de l'exemple précédent : $\mathcal{F} = \exists y_1(((\forall x_1(\neg x_1 \vee y_1)) \vee ((\neg x_1) \vee y_1)) \wedge (y_1 \vee ((\neg x_1) \wedge z_1)))$ La première occurrence de y_1 dans $\mathcal{F}$ est quantifiée, la seconde occurrence de x_1 dans $\mathcal{F}$ est liée et la troisième occurrence est libre. x_1 et z_1 sont les variables libres de $\mathcal{F}$.*

Définition 53 ***(forme polie)*** *Une QBF $\mathcal{F}$ est dite polie si est seulement si chaque occurrence liée d'une variable x dans $\mathcal{F}$ est dans la portée d'un unique quantificateur et chaque variable liée n'a pas d'occurrence liée.*

Définition 54 ***(forme prénexe)*** *Une QBF $\mathcal{F}$ est dite sous forme prénexe si elle est sous la forme $Q_1x_1 \ldots \mathcal{Q}_nx_n\mathcal{M}$ où $Q_i \in \{\forall, \exists\}$ et $\mathcal{M}$ une formule non quantifiée (formule propositionnelle).*

Définition 55 ***(QBF fermée)*** *Une QBF $\mathcal{F}$ est dite fermée si et seulement si elle ne contient pas de variable libre.*

Pour une formule booléenne quantifiée $\mathcal{F}$ polie, prénexe et fermée qui s'écrit sous la forme : $\mathcal{F} = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \dots \mathcal{Q}_n X_n \mathcal{M}$.

- $\mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \dots \mathcal{Q}_n X_n$ est appelé le *préfixe*
- $\mathcal{M}$ est appelée sa *matrice*.

Parfois on notera $\mathcal{Q}X\mathcal{M}$ pour désigner la QBF $\mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \dots \mathcal{Q}_n X_n \mathcal{M}$

Exemple 13 *La formule QBF suivante est polie, prénexe et fermée.*

$$\mathcal{F} = \exists y_1 \exists y_2 \forall x_1 \forall x_2 \exists y_3 \left\{ \begin{array}{lll} (y_2 \vee x_1 \vee \neg x_2 \vee y_3) & \wedge & (x_2 \vee x_2 \vee \neg y_3) \quad \wedge \\ (\neg y_1 \vee \neg y_2 \vee \neg y_3) & \wedge & (y_2 \vee \neg y_3) \quad \wedge \\ (\neg y_1 \vee x_2 \vee y_3) & \wedge & (\neg x_1 \vee y_2 \vee \neg x_1) \end{array} \right.$$

Étant donnée une formule QBF $\mathcal{F}$ polie, prénexe et fermée, nous pouvons grouper sous le même quantificateur l'ensemble des variables qui se suivent dans le préfixe et dont les quantificateurs associés sont identiques. Le préfixe de l'exemple 13 pourra donc être écrit sous la forme $\exists y_1 y_2 \forall x_1 x_2 \exists y_3$. Dans ce cas le préfixe de $\mathcal{F}$ sera une alternance de quantificateurs existentiels et universels portant sur un sous-ensemble disjoint de variables ($\mathcal{F}$ est polie) :

1. $\mathcal{Q}_i \in \{\forall, \exists\}$
2. $\mathcal{F} = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \dots \mathcal{Q}_n X_n \mathcal{M}$
3. $\mathcal{V}_{X_i} \cap \mathcal{V}_{X_j} = \emptyset$ tel que $(i, j) \in [1 \dots n]^2$ et $i \neq j$
4. $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$, $i \in [1 \dots n-1]$

Chaque quantification de la forme $\mathcal{Q}_i X_i$ est appelée *groupe de quantificateur* ou quantificateur tout court. $\mathcal{Q}_1$ est appelé le *quantificateur externe* et $\mathcal{Q}_n$ le *quantificateur interne*.

On définit $\mathcal{V}(\mathcal{F}) = \bigcup_{i \in \{1,\dots,n\}} X_i$ l'ensemble des variables de $\mathcal{F}$. Pour une variable $x \in \mathcal{V}(\mathcal{F})$ tq. $x \in X_k$, on définit $rang(x) = k$ et $q(x) = \mathcal{Q}_k$ désigne son quantificateur associé. Les variables apparaissant dans le même groupe de quantificateur ont le même rang.

L'ensemble des variables quantifiées universellement de $\mathcal{F}$ est désigné par $\mathcal{V}^{\forall}(\mathcal{F}) = \{x | x \in \mathcal{V}(\mathcal{F}),\ q(x) = \forall\}$ et celui des variables quantifiées existentiellement est noté $\mathcal{V}^{\exists}(\mathcal{F}) = \{x | x \in \mathcal{V}(\mathcal{F}),\ q(x) = \exists\}$

Définition 56 ***(kQBF)*** *Soit k un entier. Une kQBF est une QBF dans laquelle les quantificateurs sont appliqués à k ensembles disjoints de variables et dans laquelle les quantificateurs sont alternés. Parfois, nous ajoutons un indice dénotant le type du quantificateur le plus externe de la formule.*

Exemple 14 ***(3QBF)***

Soient X_1, X_2 et X_3 des ensembles disjoints de variables d'une formule propositionnelle $\mathcal{M}$ tels que $\mathcal{V}(\mathcal{M}) = X_1 \cup X_2 \cup X_3$. La formule $\exists X_1 \forall X_2 \exists X_3 \mathcal{M}$ est une $3QBF$.

Définition 57 (QCNF) *Une formule $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ est dite sous forme QCNF si sa matrice est sous forme normale conjonctive.*

Définition 58 *Une variable x est dite* maximale *(resp.* minimale*) dans un ensemble $\mathcal{S}$ si $x \in \mathcal{S}$ et $\forall y \in \mathcal{S}$, on a $rang(y) \leq rang(x)$ (resp. $rang(y) \geq rang(x)$).*

Définition 59 (Littéral existentiel, universel, monotone) *Soit $F = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ une formule QBF et $\mathcal{V}(\mathcal{F})$ l'ensemble de ces variables. $\mathcal{L}(\mathcal{F}) = \bigcup_{i \in \{1,\dots,n\}} \mathcal{L}(X_i)$ l'ensemble total des littéraux de $\mathcal{F}$, où $\mathcal{L}(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$.*

- *un littéral est dit* existentiel *(respectivement* universel*) ssi sa variable correspondante est quantifiée existentiellement (respectivement universellement).*
- *Un littéral x est dit* monotone *si est seulement si $\neg x$ n'apparaît pas dans la matrice de $\mathcal{F}$*

6.2 Sémantique

Définition 60 (affectation) *Pour une QBF $\mathcal{F} = \mathcal{Q}X\mathcal{M}$, et un littéral $x \in \mathcal{V}(\mathcal{F})$, l'affectation du littéral x noté $\mathcal{F}|_x$ consiste à supprimer de $\mathcal{M}$ toutes les clauses contenant une occurrence de x, à supprimer $\neg x$ de toutes les clauses où il apparaît et à éliminer la variable $|x|$ de son groupe de quantificateur associé.*

Pour un ensemble de littéraux $\{x_1, x_2 \dots x_n\}$, $\mathcal{F}|_{\{x_1,x_2\dots,x_n\}}$ désigne la formule QBF réduite par l'affectation de l'ensemble $\{x_1, x_2 \dots x_n\}$.

Exemple 15 *Considérons la QBF suivante :*

$$\mathcal{F} = \exists y_1 \forall x_1 \exists y_2 \forall x_2 \exists y_3 \left\{ \begin{array}{l} (y_1 \vee x_1 \vee \neg y_2 \vee \neg y_3) \wedge \\ (y_2 \vee x_2 \vee y_3) \end{array} \right.$$

$\mathcal{F}|_{x_1} = \exists y_2 \forall x_2 \exists y_3 (y_2 \vee x_2 \vee y_3)$.

La notion de validité d'une formule QBF est plus complexe à définir que celle d'une formule propositionnelle. En effet, la validité (satisfiabilité) d'une formule QBF consiste à trouver un ensemble de modèles propositionnels de sa matrice satisfaisant certaines conditions. Cet ensemble est appelé politique totale [COSTE-MARQUIS *et al.* 2006]. Elle est décrite récursivement dans la définition 61, mais il est nécessaire d'introduire auparavant quelques notations.

Soit S l'ensemble des affectations sur l'ensemble des variables $\mathcal{V}(\mathcal{F})$. La In-projection (resp. Out-projection) d'un ensemble d'affectations S sur un ensemble de variables $X \subseteq \mathcal{V}(\mathcal{F})$, est dénotée par $S[X]$ (resp. $S \backslash X$). Elle est obtenue en restreignant chaque affectation aux littéraux appartenant à X (resp. dans $\mathcal{V}(\mathcal{F}) \backslash X$). L'ensemble des affectations possibles sur X est dénoté par 2^X. Une affectation des variables de X est dénotée par l'ensemble de littéraux $\overrightarrow{x}$. De la même manière, la In-projection et la Out-projection peuvent aussi s'appliquer sur l'ensemble des littéraux $\overrightarrow{x}$. Si $\overrightarrow{y}$ est une affectation des variables de Y tq. $Y \cap X = \emptyset$, alors $\overrightarrow{y}.S$ décrit l'ensemble d'interprétations obtenu en concaténant $\overrightarrow{y}$ avec chaque interprétation de S. Finalement, $\mathcal{M}(\overrightarrow{x})$ dénote la formule $\mathcal{M}$ simplifiée par l'affectation partielle $\overrightarrow{x}$.

Définition 61 (politique totale) *Soit $\mathcal{F} = \mathcal{Q}_1 X_1, \dots, \mathcal{Q}_n X_n \mathcal{M}$ une formule booléenne quantifiée et $\pi = \{m_1, \dots, m_n\}$ un ensemble de modèles de sa matrice $\mathcal{M}$. π est une politique totale de la QBF $\mathcal{F}$ si et seulement si π vérifie récursivement les conditions suivantes :*

1. *$n = 0$, et $\mathcal{M} = \top$*
2. *si $\mathcal{Q}_1 = \forall$, alors $\pi[X_1] = 2^{X_1}$, et $\forall \overrightarrow{x}_1 \in 2^{X_1}$, $\pi \backslash \overrightarrow{x}_1$ est une politique totale de $\mathcal{Q}_2 X_2, \dots, \mathcal{Q}_n X_n \mathcal{M}(\overrightarrow{x}_1)$*
3. *si $\mathcal{Q}_1 = \exists$, alors $\pi[\overrightarrow{X}_1] = \{\overrightarrow{x}_1\}$ et $\pi \backslash \overrightarrow{x}_1$ est une politique totale de $\mathcal{Q}_2 X_2, \dots, \mathcal{Q}_n X_n \mathcal{M}(\overrightarrow{x}_1)$*

Remarque 4 *Soit π une politique totale de $\mathcal{F} = \mathcal{Q}_1 X_1, \ldots, \mathcal{Q}_n X_n \mathcal{M}$. Si $\mathcal{Q}_1 = \forall$ alors on peut réécrire π comme $\bigcup_{\overrightarrow{x}_1 \in 2^{X_1}} \{\overrightarrow{x}_1.(\pi \backslash \overrightarrow{x}_1)\}$ et si $\mathcal{Q}_1 = \exists$, alors $\pi[X_1] = \{\overrightarrow{x}_1\}$ et π peut être réécrite sous forme $\{\overrightarrow{x}_1.(\pi \backslash \overrightarrow{x}_1)\}$*

Problème QBF

Définition 62 (problème QBF) *Soit $\mathcal{F}$ une formule QCNF. Le problème QBF est un problème de décision qui consiste à déterminer si $\mathcal{F}$ admet une politique totale.*

Exemple 16 *Considérons la QBF suivante :*

$$\mathcal{F} = \exists y_1 \forall x_1 \exists y_2 \forall x_2 \exists y_3 \left\{ \begin{array}{l} (y_1 \vee x_1 \vee \neg y_2 \vee \neg y_3) \wedge \\ (y_2 \vee x_2 \vee y_3) \end{array} \right.$$

La figure 6.1 montre une politique satisfaisant la formule $\mathcal{F}$

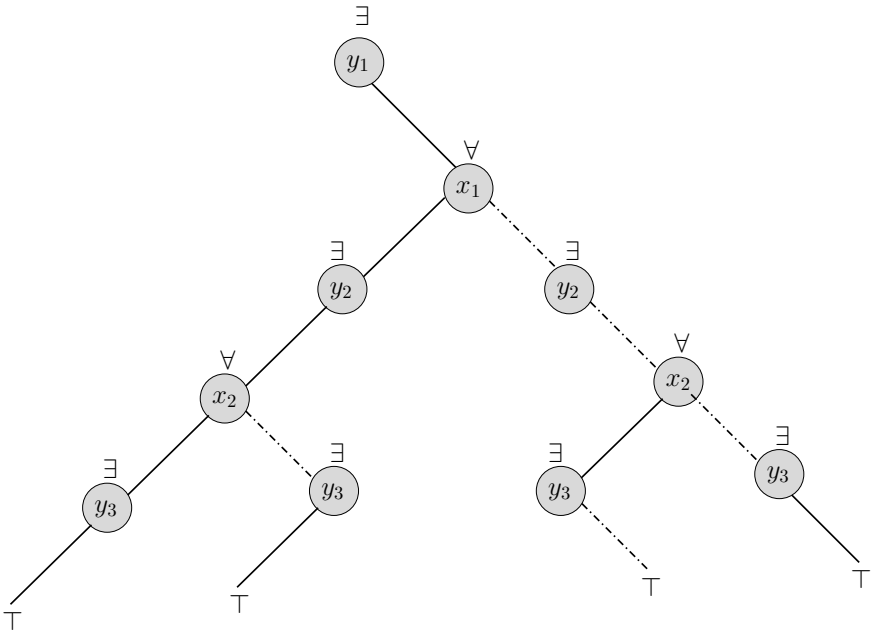


FIG. 6.1 – Politique totale de la QCNF $\mathcal{F}$

6.3 Règles de simplification

Comme dans le cas SAT, il existe plusieurs opérateurs visant à simplifier les formules booléennes quantifiées. La majorité de ces simplifications est opérée sur des formules QBF dont la matrice est sous forme normale conjonctive. Dans la suite, seules les QCNF sont considérées. Quelques unes de ces simplifications visent à réduire la base des clauses, en éliminant les littéraux universels maximaux, les littéraux monotones ou encore en inversant des quantificateurs.

Réduction universelle

Définition 63 ***réduction-universelle***
Soit $\mathcal{F} = \mathcal{Q} X \mathcal{M}$ une formule QCNF et c une clause de $\mathcal{M}$ telle que :

1. $x \in c$ *et* $q(x) = \forall$
2. $\forall y \in c$, $q(y) = \exists$ *on a* $rang(y) < rang(x)$

alors x *peut être supprimé de* c *en préservant l'équivalence avec la formule originale. Cette opération est appelée* réduction-universelle.

Propriété 16 *Soient $\mathcal{F}$ formule booléenne quantifiée et $\mathcal{F}'$ la formule obtenue en appliquant la réduction-universelle aux clauses de $\mathcal{F}$. $\mathcal{F}$ et $\mathcal{F}'$ sont équivalentes.*

En appliquant la réduction universelle à toute formule QCNF $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$, on peut déduire que le quantificateur le plus interne Q_n peut être considéré comme existentiel $Q_n = \exists$.

Q-résolution

La Q-résolution, décrite dans [BÜNING *et al.* 1995] par H. Kleine Büing, M. Karpinski et A. Flogel, est une adaptation aux QBFs du célèbre principe de résolution de Robinson [ROBINSON 1965].

La *Q-résolution* est définie par l'application de la résolution classique sur deux clauses contenant deux littéraux existentiels complémentaires suivie d'une réduction-universelle sur la résolvante générée. La Q-résolution suffit à prouver la validité ou la non validité d'une formule QBF.

Comme pour le cas SAT, la Q-résolution est difficile à mettre en œuvre en pratique. Cependant, il existe des variantes de résolution qui ont été adaptées au cas QBF comme l'hyper-résolution binaire [SAMULOWITZ & BACCHUS 2006].

Littéraux Monotones

Dans le problème SAT, un littéral monotone x peut être affecté à $vrai$ pour simplifier la formule en préservant la validité de la formule CNF associée. Ce processus peut être réitéré durant la recherche. En effet, une affectation partielle des variables de la formule peut engendrer l'apparition de littéraux monotones. L'exploitation dynamique des littéraux monotones n'est pas appliquée dans le cadre du problème SAT. En effet, aucune évaluation expérimentale n'a montré son efficacité [GIUNCHIGLIA *et al.* 2004]. Lorsqu'il s'agit des formules booléennes quantifiées, la propagation des littéraux monotones peut être bénéfique en dépit du coût de la détection [GIUNCHIGLIA *et al.* 2004, CADOLI *et al.* 1998]. Ceci peut être expliqué pour les raisons suivantes. Dans le contexte des QBF, les variables universelles augmentent considérablement la taille de l'espace à considérer. Si les littéraux monotones permettent de couper des branches qui devaient être explorées, on peut éventuellement penser que ce coût sera amorti. En outre, les littéraux monotones sont traités différemment que dans le cas SAT. L'affectation des littéraux monotones existentiels est similaire au cas SAT. Au contraire, pour un littéral universel monotone x, l'affectation est faite sur la décision qui contraint le plus la formule, à savoir $\neg x$.

Exemple 17 *Considérons la QBF suivante.*

$$\mathcal{F} = \exists y_1 y_2 \forall x_1 x_2 \exists y_3 y_4 \left\{ \begin{array}{llllll} (x_1 \vee \neg x_2) & \wedge & (y_1 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1 \vee y_4) & \wedge & (\neg y_1 \vee \neg y_3) & \wedge \\ (\neg y_1 \vee y_3) & \wedge & (y_1 \vee x_2 \vee x_3 \vee \neg y_4) & \wedge \\ (y_2 \vee x_1 \vee x_2 \vee y_4) \end{array} \right.$$

le littéral y_2 est monotone dans $\mathcal{F}$. La formule $\mathcal{F}'$ obtenue après simplification est la suivante :

$$\mathcal{F} = \exists y_1 \forall x_1 x_2 \exists y_3 y_4 \left\{ \begin{array}{llllll} (x_1 \vee \neg x_2) & \wedge & (y_1 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1 \vee y_4) & \wedge & (\neg y_1 \vee \neg y_3) & \wedge \\ (\neg y_1 \vee y_3) & \wedge & (y_1 \vee x_2 \vee x_3 \vee \neg y_4) & \wedge \end{array} \right.$$

Fausseté & vérité triviales

La fausseté et la vérité triviales référencées dans [GIUNCHIGLIA *et al.* 2001b] sont deux approches de pré-traitement permettant par simples opérations sur la formule de répondre à la question de la (non)validité d'une formule QBF. Ces deux approches prennent en compte la composition de la formule et essayent de la diviser en sous-parties afin de trouver des conditions suffisantes pour la (non)validité de la QBF. Bien que ces deux approches puissent être utilisées durant la recherche, elles ne sont pas actuellement intégrées dans les solveurs QBF.

Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ une QBF. On peut partitionner $\mathcal{M}$ en trois sous-formules :
- $\mathcal{E}(\exists)$ contient les clauses de $\mathcal{M}$ dans lesquelles seules les occurrences de $\mathcal{V}^{\exists}(\mathcal{F})$ apparaissent
- $\mathcal{U}(\forall)$ contient les clauses de $\mathcal{M}$ dans lesquelles seules les occurrences de $\mathcal{V}^{\forall}(\mathcal{F})$ apparaissent
- $\mathcal{R}(\exists, \forall)$ contient le reste des clauses.

La formule $\mathcal{F}$ est réécrite sous la forme : $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_k X_n \; \mathcal{E}(\exists) \wedge \mathcal{U}(\forall) \wedge \mathcal{R}(\exists, \forall)$

Exemple 18 *Considérons la formule de l'exemple 17.*
On a :

$$\mathcal{E} = \left\{ \begin{array}{l} (\neg y_1 \vee y_3) \\ (y_1 \vee y_3) \\ (\neg y_1 \vee \neg y_3) \end{array} \right.$$

$$\mathcal{U}(\forall) = \left\{ \quad (x_1 \vee \neg x_2) \right.$$

$$\mathcal{R}(\exists, \forall) = \left\{ \begin{array}{l} (y_1 \vee \neg x_1 \vee y_4) \\ (y_1 \vee x_2 \vee x_3 \vee \neg y_4) \\ (y_2 \vee x_1 \vee x_2 \vee y_4) \end{array} \right.$$

Propriété 17 (fausseté triviale sur $\mathcal{V}^{\forall}(\mathcal{F})$) *Une* QBF *est non-valide si $\mathcal{U}'(\forall) \neq \emptyset$ où $\mathcal{U}'(\forall)$ est obtenue en supprimant de $\mathcal{U}(\forall)$ toutes les clauses tautologiques.*

Cette propriété est assez simple à vérifier. En effet si $\mathcal{U}'(\forall) \neq \emptyset$, par application de la réduction-universelle, toute clause de $\mathcal{U}'(\forall)$ est réduite à la clause vide $\perp$.

Propriété 18 (fausseté triviale sur $\mathcal{V}^{\exists}(\mathcal{F})$) *Une formule booléenne quantifiée $\mathcal{F}$ est insatisfaisable si $\mathcal{E}(\exists)$ est insatisfaisable.*

Cette propriété constitue une condition suffisante pour la non validité d'une QBF.

Propriété 19 (vérité triviale) *Une QBF $\mathcal{F}$ est valide si $\mathcal{E}(\exists) \wedge \mathcal{R}'(\exists, \forall)$ est satisfiable où $\mathcal{R}'(\exists, \forall)$ est obtenue en supprimant de $\mathcal{R}(\exists, \forall)$ tous les littéraux universels.*

Exemple 19 *pour la formule de l'exemple 17,*

$$\mathcal{E}(\exists) \wedge \mathcal{R}'(\exists, \forall) = \left\{ \begin{array}{llll} (y_1 \vee y_4) & \wedge & (\neg y_1 \vee y_3) & \wedge \\ (y_1 \vee \neg y_4) & \wedge & (y_1 \vee y_3) & \wedge \\ (y_2 \vee y_4) & \wedge & (\neg y_1 \vee \neg y_3) & \wedge \end{array} \right.$$

L'interprétation $\{y_1, y_2, y_3, y_4\}$ est un modèle de $\mathcal{E}(\exists) \wedge \mathcal{R}'(\exists, \forall)$. Par conséquent, la formule QBF $\mathcal{F}$ est valide.

Inversion de quantificateurs

Dans [RINTANEN 1999, RINTANEN 2001], J. Rintanen met en œuvre une méthode d'élagage de l'espace de recherche exploré par les solveurs QBF basés sur DPLL : l'inversion de quantificateurs. L'idée est la suivante : étant donnée une QBF de la forme $\exists Y_1 \forall X_1 \mathcal{F}$, il est intéressant de considérer également la QBF $\forall X_1 \exists Y_1 \mathcal{F}$ qui est une conséquence logique de $\exists Y_1 \forall X_1 \mathcal{F}$. En réalité seule la contraposée nous intéresse. En effet, si $\exists Y_1 \forall X_1 \mathcal{F} \models \forall X_1 \exists Y_1 \mathcal{F}$, alors $\neg(\forall X_1 \exists Y_1 \mathcal{F}) \models \neg(\exists Y_1 \forall X_1 \mathcal{F})$. Ainsi pour certaines valeurs des variables de X_1, il n'existe pas de valeurs possibles pour Y_1 qui préservent la validité de $\forall X_1 \exists Y_1 \mathcal{F}$. Ces mêmes valeurs falsifient $\exists X_1 \forall Y_1 \mathcal{F}$. De cette manière, on évite d'explorer de mauvaises branches.

6.4 Algorithmes de résolution pour QBF

La plupart des solveurs QBF, (e.g.,[LETZ 2002b, GIUNCHIGLIA *et al.* 2001d]) sont des adaptations de la procédure DPLL au QBFs. Toutefois, d'autres approches existent. Elles sont généralement basées sur les techniques de transformation comme la Q-résolution ou la skolemization. Lorsque l'on passe du problème SAT au problème QBF, deux difficultés principales s'ajoutent au problème. La première est l'universalité d'un sous-ensemble de variables qui nous oblige à considérer pour chacune d'entre elles les deux valeurs de vérité possibles. Ce qui induit une augmentation de la taille de l'espace de recherche. La seconde réside dans l'ordre partiel des variables imposé par le préfixe. Cet ordre rajoute une complexité supplémentaire à l'efficacité des heuristiques de choix de variable. La première propriété (universalité) est liée intrinèquement aux QBFs et doit donc être adoptée d'une certaine façon par tous les solveurs QBF. La seconde propriété peut être contournée, et diverses tentatives ont été menées pour s'affranchir de l'ordre partiel sur les variables. Dans le suite, nous présentons tout d'abord un algorithme de recherche pour les formules booléennes quantifiées. Cet algorithme est une procédure DPLL pour SAT légèrement modifiée pour le cas QBF. Cet algorithme est amélioré dans la section 2.3 grâce à l'introduction de l'apprentissage, composant essentiel dans les solveurs QBF modernes. Ensuite nous présentons les principales techniques de transformation.

6.4.1 DPLL pour QBF

Comme décrit dans le chapitre 1, une procédure DPLL classique est une fonction récursive qui progresse par affectation de variables (décision et propagation unitaire) et implémente un mécanisme classique de retour-arrière en cas d'échec. L'algorithme 9 pour QBF fonctionne sur le même principe. Il reçoit en entrée une formule QBF et progresse par affectations de variables. Le choix de ces variables est partiellement déterminé par l'ordre du préfixe mais pour l'ensemble des variables d'un même groupe de quantificateurs, le choix peut être effectué via une heuristique. L'algorithme, commence par affecter une variable du groupe de quantificateurs le plus externe (ligne 4), propage ensuite les clauses unitaires. En cas de conflit, deux cas sont à distinguer. Si la dernière affectation est une variable existentielle, un retour-arrière est effectué pour tester l'autre valeur de vérité (ligne 6). Dans le cas d'une variable universelle, si un modèle est trouvé, alors l'algorithme effectue un retour-arrière pour tester l'autre valeur de vérité (ligne 7), sinon, le test de l'autre valeur de vérité n'est pas effectué. Notons qu'en cas de retour-arrière chronologique à partir d'un conflit, la variable de branchement ne peut être qu'existentielle. Cependant en cas de retour à partir d'un succès (solution trouvée), les variables de branchement sont des variables universelles.

Algorithm 9: Algorithme $\mathcal{Q} - \mathcal{DPLL}$ pour QBF

Input: une QCNF $\mathcal{F}$;
Output: vrai si $\mathcal{F}$ est satisfiable, faux sinon;

```
1 begin
2     if (F == ∅) then return vrai ;        // retourner vrai si la QBF est vide
3     if (⊥ ∈ F) then return faux ;           // contient une clause falsifié
4     Choisir une variable x ;                // quantificateur le plus externe
5     Résultat = Q − DPLL(F|_x)
6     if (q(x) = ∃ et Rultat = faux) then Résultat = Q − DPLL(F|_¬x)
7     if (q(x) = ∀ et Rsultat = vrai) then Résultat = Q − DPLL(F|_¬x)
8 end
```

6.4.2 Apprentissage

Les solveurs QBF modernes utilisent deux schémas de retour-arrière non chronologique : l'analyse de conflits et l'analyse de solution. Si le premier est similaire au cas SAT, le deuxième est une nouvelle technique propre au cas QBF qui est décrite plus en détail dans la suite. Premièrement nous introduisons une extension de l'algorithme 9 qui est complet pour la recherche. L'extension est basée sur le mécanisme d'apprentissage des conflits et des solutions. La technique $\mathcal{Q} - CSDCL$ (*"Conflict and Solution Driven Clause Learning"*) résultante est décrite par l'algorithme 10. L'analyse de conflits permet d'extraire de nouvelles clauses permettant d'éviter les interprétations déjà prouvées invalides. L'analyse de solutions permet, quant à elle, de ne pas re-visiter les mêmes modèles déjà découverts. L'algorithme 10 est plus sophistiqué dans la mesure où l'apprentissage est fait à la fois à partir des conflits et des modèles trouvés. Les solveurs QBF modernes opèrent sur deux formules, l'une sous forme normale conjonctive représentant les clauses apprises à partir des conflits et l'autre sous forme normale disjonctive représentant les cubes obtenus lors des analyses de solutions. L'analyse de conflits et des solutions ouvre de nouvelles perspectives dans la mesure où elle permet de réduire la taille de l'espace de recherche grâce aux informations (clauses et cubes) collectées.

Analyse de Conflits

L'analyse de conflits est une technique classique qui, via une technique de résolution permet de générer de nouvelles clauses dites clauses apprises. Cette résolution est effectuée en utilisant le graphe d'implication représentant un ensemble de littéraux à l'origine du conflit et leurs implications. La discussion détaillée de cette technique est décrite dans le chapitre 2. Cette technique permet d'identifier, pour l'interprétation courante, l'origine du conflit.

Dans le cas QBF, cette analyse diffère légèrement du cas SAT à cause des variables universelles. Deux points principaux sont à distinguer. Le premier concerne le cas où l'analyse de conflit produit comme littéral assertif un littéral universel. Dans ce cas, un premier retour-arrière est effectué au premier niveau du littéral assertif, l'analyse du conflit est réitérée jusqu'à l'obtention d'un littéral assertif existentiel. Pour le cas d'un littéral assertif existentiel, un seul retour-arrière est effectué, et la recherche continue.

Le deuxième point principal concerne l'analyse elle même. En effet, dans le cas SAT, les clauses utilisées dans le schéma d'apprentissage codent des implications avec un seul littéral à vrai. Cette caractéristique garantit que la clause assertive obtenue ne peut être tautologique. Au contraire dans le cas QBF, il arrive que la clause assertive soit tautologique. Pour s'en convaincre, considérons une formule QBF $\mathcal{F} = \mathcal{Q}X\mathcal{M}$ satisfaisant les deux conditions suivantes :

– $\exists\{c_1, c_2\} \in \mathcal{M}$ tel que $c_1 = (y_1 \vee x_1 \vee y_2)$ et $c_2 = (\neg y_1 \vee \neg x_1 \vee y_2)$
– le préfixe de $\mathcal{F}$ est de la forme $\mathcal{Q}X = (\ldots \exists y_1 \forall x_1 \exists y_2 \ldots)$.

Supposons qu'à un niveau n donné, une décision a propagé $\neg y_2$. Dans ce cas x_1 est propagé (clause c_1, grâce à la réduction-universelle) et la clause c_2 se trouve falsifiée. Pour parcourir le graphe d'implication, le schéma de résolution doit opérer une résolution entre c_1 et c_2. Or cette résolvante est tautologique. Pour contourner ce problème des adaptations sont donc nécessaires. Une première réponse consisterait à abandonner le schéma d'apprentissage et à effectuer un simple retour-arrière chronologique. Dans [ZHANG & MALIK 2002b], les auteurs proposent une définition de *longue distance* semblable au schéma d'analyse de conflit mais qui permet de prendre en compte l'apparition de clauses tautologiques.

Analyse de Solutions

Les retours-arrière à partir des solutions (ou succès) sont des conséquences de l'analyse de solutions. L'analyse de solution (e.g., [ZHANG & MALIK 2002a, GIUNCHIGLIA *et al.* 2006, LETZ 2002a]) est une technique propre au cas QBF. Elle est déclenchée lorsqu'un modèle de la matrice est détecté. Bien que la notion de cube a été déjà étudiée avant dans plusieurs travaux [ZHANG & MALIK 2002b, GIUNCHIGLIA *et al.* 2006, LETZ 2002a], nous donnons ici une formalisation de ce concept.

Commençons par définir l'opérateur $\leq_u$ entre deux ensembles consistants de littéraux. Un ensemble c de littéraux est dit consistant si $\forall l$ tel que $l \in c$ alors $\neg l \notin c$. On dit que $c' \leq_u c$ si et seulement si $c \subseteq c'$ (c' est un sur-ensemble de c) et $\forall l \in c' \backslash c$ on a $q(l) \leq umax(c)$ où $umax(c)$ est le quantificateur universel de $\mathcal{F}$ le plus interne contenant un littéral de c.

Définition 64 (CUBE) *On appelle* CUBE *d'une formule QBF $\mathcal{F}$, un ensemble consistant de littéraux c tel que pour tout ensemble c' avec $c' \leq_u c$ on a $\mathcal{F}|_{c'}$ est valide.*

Propriété 20 *Les propriétés suivantes sont vérifiées :*
– *Si c est un ensemble de littéraux satisfaisant chaque clause de $\mathcal{F}$, alors c est un cube.*
– *Si c est un cube et e un littéral existentiel maximal dans c alors $c' = c\backslash\{e\}$ est un cube.*
– *Si*
  1. *c_1 et c_2 sont deux cubes tels qu'il existe un unique littéral universel u avec $u \in c_1$ et $u \in c_2$, et*
  2. *u est maximal dans c_1 et $\neg u$ est maximal dans c_2,*

  alors $c_1 \cup c_2\backslash\{u, \neg u\}$ est un cube.

La propriété 20 capture l'essentiel du processus d'apprentissage de solutions appliqué durant la recherche. Cela est illustré dans l'exemple suivant. Supposons qu'on dispose de la formule QBF suivante :

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \exists y_3 (\neg y_1 \vee y_3) \wedge (\neg y_2 \vee x_2 \vee y_3) \wedge (\neg y_2 \vee x_1) \wedge (\neg y_3 \vee \neg x_1)$$

Soit ρ une affectation (interprétation) satisfaisant toutes les clauses de $\mathcal{F}$. ρ affecte les valeurs suivantes aux variables de $\mathcal{V}(\mathcal{F})$:

$$\rho(y_1) = faux, \rho(x_1) = faux, \rho(y_2) = faux, \rho(x_2) = vrai, \rho(y_3) = faux.$$

En considérant le premier point de la propriété 20, ρ est un cube. Cependant, étant donné un cube ρ, différents cubes valides peuvent être extraits de ρ. Ceci est dû au fait qu'une clause peut être satisfaite par

Algorithm 10: Algorithme $\mathcal{Q} - \mathcal{CDCSL}$ avec apprentissage

Input: une QCNF $\mathcal{F}$;
Output: vrai si $\mathcal{F}$ est satisfiable, faux sinon;

```
1  begin
2      simplifier(F);
3      if (⊥ ∈ F) then
4          return faux ;                          // clause vide appartient à F
5      end
6      if (F == ∅) ;                  // toutes les clauses sont satisfaites
7      then
8          return vrai
9      end
10     conflit = null;
11     solution = null;
12     while (1) do
13         if (conflit != null) ;                              // cas(1) conflit
14         then
15             (assert_level, clause[y]) = analyze_conflit();
16             if (clause[y] == ∅) then
17                 return faux ;
18             end
19             backtrack(btlevel) ;                // backtrack au niveau assertif
20             ajouter_clause(clause[y]);
21             propager(y) ;                        // y est le littéral assertif
22         end
23         if (solution != null) ;                             // cas(2) solution
24         then
25             (assert_level, cube[y]) = analyze_solution();
26             cube[z] = cube[y] ∪ cube[¬y]\{y, ¬y} ;
27             ajouter_cube(cube[z]) ;          // z littéral minimal dans cube[z]
28             if (cube[z] == ⊥) then
29                 return vrai ;
30             end
31             backtrack(assert_level);         // backtrack au niveau assertif
32             propager(¬z);
33         end
34         else
35             ChoisirVariable(x) ;                  // cas(3) nouvelle décision
36             propager(x);
37         end
38     end
39 end
```

plusieurs littéraux du modèle. Dans ce cas on peut extraire un impliquant premier ρ_{IP}, i.e, un cube de $\mathcal{F}$ tel que chaque sous-ensemble de ρ_{IP} n'est pas un cube de $\mathcal{F}$. Un modèle ρ peut admettre plusieurs impliquants. L'algorithme 10 calcule donc l'un d'entre eux et le mémorise dans la base des solutions (ligne 19). En pratique, il est plus intéressant d'éliminer les littéraux universels minimaux (les plus internes) car cela permet un retour-arrière plus intéressant dans l'arbre de recherche.

Dans le cas général, les solveurs essayent de minimiser à la fois le nombre total des littéraux du cube et le nombre des littéraux universels ([SAMULOWITZ & BACCHUS 2006]) apparaissant dans le cube. Plus le nombre de littéraux universels dans un cube est petit, plus le nombre d'affectations à vérifier pour les littéraux universels l'est aussi. Minimiser le nombre total des littéraux du cube et le nombre des littéraux universels revient à maximiser le nombre des littéraux existentiels maximaux.

Cette méthode est motivée en pratique par le fait que les littéraux existentiels maximaux satisfont plus de clauses.(e.g., [SAMULOWITZ & BACCHUS 2006, SAMULOWITZ & BACCHUS 2005]) et peuvent donc être supprimés du cube (deuxième point de la propriété 20). Dans l'exemple précédent, l'interprétation $\{x_1, y_2, x_2, \neg y_3\}$ satisfait la matrice de $\mathcal{F}$. Après élimination des littéraux existentiels maximaux, on obtient le cube réduit suivant : $c_1 = \{x_1, y_2, x_2\}$. Après le calcul d'un premier cube c_1, et l'élimination des littéraux existentiels minimaux, l'algorithme effectue un retour-arrière sur la variable universelle la plus interne de c, en sautant les différentes variables intermédiaires du préfixe qui ne sont pas mentionnées dans le cube.

Comme montré dans la propriété 20 (troisième point) un cube contenant une affectation d'un littéral universel x peut être combiné avec un autre cube contenant son complémentaire $\neg x$ pour obtenir un nouveau cube dans un processus proche de la résolution de clauses (ligne 24). En particulier, si le littéral universel le plus interne dans le cube a déjà son autre valeur de vérité prouvée vraie (un autre modèle a été trouvé avec l'autre valeur de vérité), le solveur peut combiner ces deux cubes et éliminer ce littéral. La symétrie apparente entre la résolution de clauses et celles des cubes est détaillée dans [GIUNCHIGLIA *et al.* 2006].

Revenons à notre exemple et supposons que le solveur applique un retour-arrière à partir des solutions sur le cube, i.e, $c_1 = \{x_1, y_2, x_2\}$. Le littéral universel le plus interne de la clause est x_2. Comme on doit trouver deux modèles avec les deux valeurs de vérité de x_2, le solveur défait ces décisions jusqu'à x_2. Il force ensuite, x_2 à $faux$ ($\neg x_2$) et dans ce cas il va essayer de résoudre la sous-formule $F|_{\{\neg x_1, y_2, \neg x_2\}}$. Supposons que cette nouvelle recherche aboutisse à la solution ρ' suivante :

$$\rho'(y_1) = faux, \rho'(x_1) = faux, \rho'(y_2) = faux, \rho'(x_2) = faux, \rho'(y_3) = vrai$$

A partir de cette solution ρ', on extrait un impliquant premier de la forme $c_2 = \{\neg x_1, y_2, \neg x_2\}$. A ce point, pour les deux valeurs de vérité de x_2 on a trouvé un modèle. En appliquant la résolution entre ces deux modèles c_1 et c_2, on obtient un nouveau cube $c_3 = \{\neg x_1\}$ où le littéral existentiel interne y_2 (maximale) est éliminé. Le cube c_3 exprime la validité de la sous-formule $F|_{\neg x_1}$.

Pour plus de détails le lecteur peut se référer à [GIUNCHIGLIA *et al.* 2001c, ZHANG *et al.* 2001, LETZ 2002a, GIUNCHIGLIA *et al.* 2006].

6.5 Transformations : de QBF vers SAT

Dans le cas général, toutes les autres approches pour la résolution de formules booléennes quantifiées sont basées sur l'élimination de quantificateurs. En d'autres termes, ces méthodes tentent de convertir un problème QBF en un problème SAT. L'une de ces transformations est basée sur la procédure de résolution de Davis et Putnam, et une autre est basée sur la skolemization pour obtenir une formule CNF purement

propositionnelle.

Dans un premier temps on décrit comment l'élimination de variables est utilisée pour supprimer le préfixe. Ensuite, différentes extensions de ce schéma basique sont discutées et finalement nous verrons comment transformer une QBF en une formule booléenne CNF en utilisant les fonctions de skolem.

6.5.1 Élimination de variables dans les QBF

Le schéma d'élimination de variables dans le cas QBF est similaire à l'algorithme *DP*. Cette technique restreint son application aux variables existentielles.

En effet cette méthode débute par l'élimination des variables existentielles du groupe de quantificateurs le plus interne ($Q_n = \exists$). Une fois ces variables éliminées, le quantificateur le plus interne $\mathcal{Q}_{n-1}X_{n-1}$ devient universel ($Q_{n-1} = \forall$). En appliquant la $rduction - universelle$ aux clauses de la formule obtenue, on peut tout simplement supprimer des clauses de la formule les occurrences des variables de X_{n-1}. Cette technique est appliquée jusqu'à atteindre le quantificateur le plus externe Q_1X_1 . Si ce dernier est existentiel alors la formule obtenue n'est autre qu'une simple formule propositionnelle. Dans le cas contraire une simple application de la réduction-universelle permet d'aboutir au même résultat. Cependant, comme dans le cas SAT, le nombre de clauses générées tout le long du processus d'élimination peut être exponentiel.

6.5.2 Résolution et extension

Dans [BIERE 2004], l'élimination de variables existentielles est combinée avec l'extension des variables universelles dans le but de réduire la taille de la théorie résultante. Une variable appartenant au quantificateur universel le plus interne de $QX\mathcal{F}$ est évaluée à ces deux valeurs de vérité (étape d'extension). Pour étendre une variable universelle x appartenant au quantificateur universel le plus interne $\mathcal{Q}_{n-1}X_{n-1}$, il est nécessaire de générer une copie de l'ensemble des variables existentielles qui le suit dans le préfixe $\mathcal{Q}_nX_n$. Dans le même temps, une copie de toutes les clauses contenant les variables de X_n est générée en substituant dans chacune de ces clauses chaque variable par sa copie correspondante. Soit $\mathcal{S}_1$ l'ensemble des clauses originales et $\mathcal{S}'_1$ l'ensemble de clauses contenant la copie. Donc la variable universelle x sélectionnée pour l'opération d'extension est affectée à $vrai$ dans $\mathcal{S}_1$ et à $faux$ dans $\mathcal{S}'_1$. L'étape d'extension applique la réduction suivante : $\mathcal{S}_1|_x$ et $\mathcal{S}'_1|_{\neg x}$. Par conséquent, la formule propositionnelle résultante $\mathcal{F}'$ après l'extension de x est égale à $\mathcal{F}\backslash\mathcal{S}_1 \cup (\mathcal{S}_1|_x \cup \mathcal{S}'_1|_{\neg x})$.

Enfin, la copie des variables existentielles est ajoutée au quantificateur le plus interne et la variable x est tout simplement éliminée de $\mathcal{Q}_{n-1}X_{n-1}$. Pour illustrer cette opération considérons la formule QBF suivante : $\forall x_1 \exists y_1 (x_1 \vee y_1)$. L'extension de x mène vers la nouvelle formule non simplifiée suivante :

$$\exists y_1 \exists y'_1 (vrai \vee y_1) \wedge (faux \vee y'_1).$$

Cette méthode d'extension d'une seule variable peut être appliquée à chaque variable appartenant au groupe de quantificateurs universelles le plus interne $\mathcal{Q}_{n-1}X_{n-1}$. Par conséquent, il est possible d'éliminer toutes les variables universelles en itérant l'extension dans le but d'obtenir à la fin de ce processus une simple formule booléenne. Cette dernière est satisfiable si et seulement si la formule QBF originale est satisfiable.

Quantor [BIERE 2004] est l'un des solveurs modernes basé sur le principe d'élimination et d'extension de variables ("resolve and expand"). Il utilise une heuristique sophistiquée pour alterner la résolution et l'extension de variables. Le schéma essaye d'estimer le coût de la résolution et de l'extension et applique la moins coûteuse parmi elles en terme d'espace. Dans ce but, l'heuristique est utilisée pour choisir une variable universelle à étendre ou une variable existentielle à éliminer par résolution dans le

but de minimiser la taille de la sous-formule résultante. En plus, un raisonnement sur les équivalences et la subsumption est utilisé pour minimiser encore plus la taille de la formule propositionnelle résultante. Finalement, un algorithme SAT moderne est invoqué pour résoudre la formule CNF obtenue. Sur quelques famille d'instances, Quantor s'avère très efficace. En plus, il peut résoudre des instances qui ne sont résolues par aucun des solveurs basés sur la recherche. Cependant, à l'inverse, les solveurs basés sur la recherche sont capables de résoudre des classes d'instances que Quantor n'arrive pas à résoudre (ceci est dû particulièrement à des problèmes d'espace mémoire) [NARIZZANO *et al.* 2006].

Récemment, [BUBECK & BÜNING 2007] introduit un pré-traitement basé sur Quantor. Le but principal de ce pré-traitement est de simplifier la théorie en employant la technique d'élimination de variables et l'extension des universels de telle sorte que la taille de la formule résultante soit de taille raisonnable (e.g., au plus deux fois la taille de la QBF originale). Le but du pré-traitement est donc d'éliminer le plus possible de quantificateurs afin de minimiser l'impact du préfixe et essayer de réduire la taille de l'arbre de recherche. Les résultats empiriques ne sont pas complètement convaincants en raison des résultats pas très satisfaisants sur les instances structurées. Cependant, l'approche semble prometteuse et mérite d'être explorée davantage.

6.5.3 Élimination symbolique des quantificateurs

Dans [PAN & VARDI 2004], l'élimination symbolique des quantificateurs est introduite. La technique n'est pas seulement basée sur le procédure de résolution de Davis-Putnam, mais aussi sur la représentation sous forme de $\mathcal{ZBDD}$ [ICHI MINATO 1996] des clauses de $\mathcal{F}$. Les variables sont éliminées en commençant par les plus internes. L'élimination des variables existentielles est basée sur la **multi-résolution** [CHATALIC & SIMON 2000]. La multi-résolution effectue toutes les résolutions possibles sur une variable y en une seule opération via des opérations sur la représentation sous forme $\mathcal{ZBDD}$ des clauses.

Cependant, dû à la représentation sous forme $\mathcal{ZBDD}$ le coût de l'application de la multi-résolution ne dépend pas du nombre de clauses mais plutôt de la taille du codage de ces clauses [CHATALIC & SIMON 2000].

Avec cette représentation il est possible d'opérer un grand nombre de résolutions en une seule étape. Cependant un grand nombre de résolvantes peut être produit. Toutefois, cela nécessite que le $\mathcal{ZBDD}$ sous-jacent admette une taille raisonnable. Par conséquent, si la représentation ne permet pas d'obtenir un ensemble de clauses plus compact, l'application de la multi-résolution reste inefficace [CHATALIC & SIMON 2000]. L'algorithme *QMRES* introduit dans [PAN & VARDI 2004] opère sur une représentation sous forme de $\mathcal{ZBDD}$ des clauses. En commençant par le quantificateur le plus interne $\mathcal{Q}_n$, toutes les variables existentielles X_n sont éliminées par multi-résolution. Ensuite, celles quantifiées universellement et appartenant au groupe de quantificateurs $\mathcal{Q}_{n_1}X_{n-1}$ sont supprimées de la formule en appliquant la réduction-universelle.

L'algorithme est basé principalement sur les deux opérateurs suivants : l'opérateur $\times$ qui génère à partir de deux ensembles de clauses $\mathcal{S}_1$ et $\mathcal{S}_2$ l'ensemble $\mathcal{S} = \{c | \exists c_1 \in \mathcal{S}_1, \exists c_2 \in \mathcal{S}_2, c = c_1 \cup c_2\}$.

Et l'opérateur $+$ qui, appliqué aux deux ensembles de clauses $\S_1$ et $\mathcal{S}_2$, consiste à appliquer la subsumption sur l'ensemble $\mathcal{S}$ de sorte que pour toute clause $c \in \mathcal{S}$ il n'existe pas de clause $c' \subset c$ telle que $c' \in \S$. L'opérateur $\times$ peut être utilisé pour opérer la multi-résolution sur les deux ensembles distincts de clauses.

Étant donné un $\mathcal{ZBDD}$ f, f_x^+ et f_x^- dénotent les ensembles suivants $f_x^+ = \{c | c \vee x \in f\}$ et $f_x^- = \{c | c \vee \neg x \in f\}$. $f_{\setminus\{x,\neg x\}}$ représente l'ensemble de clauses qui ne contient ni x ni $\neg x$. La procédure complète est décrite brièvement dans l'algorithme 11. Les résultats expérimentaux observés sur *QMRES* sont compétitifs sur des ensembles particuliers d'instances [NARIZZANO *et al.* 2006].

Algorithm 11: Multi-Résolution pour QBF en utilisant des $\mathcal{ZBDD}$

Input: $\mathcal{F}$ une QBF;
$\mathcal{S}$ est la représentation sous forme $\mathcal{ZBDD}$ des clauses de $\mathcal{F}$;
$\mathcal{V}$ est la liste des variables de $\mathcal{F}$ telle que $q(x_{i+1}) \leq q(x_i)$;
Output: $vrai$ si la QBF est valide;

```
1  begin
2      for (i = 1 to n) do
3          if (x_i est existentielle) then
4              S ← (S_{x_i^-} × S_{x_i^+}) + S_{x'_i} ;        // Multi-résolution sur les
               existentielles
5          end
6          else
7              S ← (S_{x_i^-} + S_{x_i^+}) + S_{x'_i} ;        // Élimination des universelles
8              S ← propagation_unitaire(S);
9          end
10     end
11     return (S ≠ ∅)
12 end
```

6.5.4 QBF et fonctions de skolem

SKIZZO [BENEDETTI 2004, BENEDETTI 2005b] est un solveur QBF basé sur la skolemization [SKOLEM 1928]. La formule QBF est étendue à une formule logique d'ordre supérieur dans le but de rendre toutes les fonctionnalités disponibles. En particulier, l'approche requiert des symboles de fonctions et des quantifications sur ces fonctions. Premièrement cette expressivité permet de se dispenser de toutes les variables existentielles qui sont remplacées par leur fonctions de skolem correspondantes. Par conséquent, la formule résultante ne contient que des quantifications universelles et les fonctions de skolem. Dans l'étape suivante, les fonctions de skolem seront à leurs tours remplacées par des conjonctions de formules propositionnelles. Ces formules propositionnelles capturent les fonctionnalités des fonctions de Skolem et codent la signification de la formule originale. Cependant, la formule propositionnelle résultante n'est pas sous forme normale conjonctive. Pour la transformer sous cette forme afin de la résoudre avec un solveur SAT standard, ceci requiert l'introduction de variables et de clauses supplémentaires. Une fois cela accompli, la QBF sera convertie avec succès à un simple problème SAT. Si la formule propositionnelle résultante est (UN)SAT cela implique que la formule originale QBF est (non)valide. Notons cependant que ce processus peut croître exponentiellement la taille de la formule SAT obtenue [BENEDETTI 2004].

Pour détailler ce processus décrit brièvement avant, on va le décrire étape par étape via un exemple. Supposons qu'on dispose de la QBF suivante :

$$\forall x_1 \forall x_2 \exists y_1 \exists y_2 (x_1 \vee x_2 \vee y_1) \wedge (\neg x_1 \vee y_2).$$

Comme mentionné auparavant, la première étape consiste à remplacer les variables quantifiées existentiellement par leurs fonctions de skolem correspondantes. Chaque variable existentielle doit être remplacée par une nouvelle et unique fonction de skolem. L'arité de chaque fonction de skolem dépend des variables universelles qui la précédent dans le préfixe. Dans notre exemple, on a deux variables existentielles. Par conséquent, on doit introduire deux nouvelles fonctions de skolem. On note par s_1 la fonction

de skolem associée à y_1 et s_2 celle associée à y_2. L'arité des deux fonctions de skolem est deux (à savoir les deux universels x_1 et x_2). L'arité de s_2 peut encore être réduite car x_1 n'apparaît pas dans les clauses où les occurrences de y_2 apparaissent. Pour plus de détails sur la façon de réduire la dimension des fonctions de skolem, le lecteur peut se référer à [BENEDETTI 2004]. Dans notre exemple donc, on obtient les deux fonctions de skolem $s_1(x_1, x_2)$ et $s_2(x_2)$. Dans l'étape suivante, on peut remplacer les occurrences des variables quantifiées existentiellement par leurs fonctions de skolem correspondantes dans la formule originale pour obtenir la formule suivante :

$$\exists s_1 \exists s_2 \forall x_1 \forall x_2 (x_1 \vee x_2 \vee s_1(x_1, x_2)) \wedge (\neg x_1 \vee s_2(x_2))$$

Notons que pour le moment on ne connaît pas exactement la forme de la fonction de skolem mais seulement son existence. La transformation de la formule précédente en une formule purement propositionnelle requiert premièrement pour chaque fonction de skolem l'addition de variables quantifiées existentiellement et de clauses. Dans le contexte de QBF, chaque fonction de skolem est paramétrée par un ensemble de variables universelles et le résultat de cette fonction est une valeur booléenne. Premièrement, on doit introduire pour chaque variable existentielle autant de nouvelles variables existentielles que le nombre de combinaison possibles des variables universelles de sa fonction de skolem correspondante. Pour chaque combinaison possible, on utilise une variable unique pour représenter la valeur de la fonction de skolem associée à cette combinaison. Revenons à notre exemple. On doit introduire quatre nouvelles variables quantifiées existentiellement pour s_1 et deux existentielles pour s_2 :

valeurs de la fonction s_1	**valeurs de la fonction** s_2
$y^1_{s_1} \equiv s_1(x_1, x_2)$	$y^1_{s_2} \equiv s_2(x_1)$
$y^2_{s_1} \equiv s_1(\neg x_1, x_2)$	$y^2_{s_2} \equiv s_2(\neg x_1)$
$y^3_{s_1} \equiv s_1(x_1, \neg x_2)$	
$y^4_{s_1} \equiv s_1(\neg x_1, \neg x_2)$	

Notons que chaque nouvelle variable existentielle introduite est ajoutée au groupe de quantificateurs le plus externe.

Maintenant, on va présenter chaque fonction de skolem par une formule propositionnelle en utilisant les nouvelles variables additionnelles. Dans la formule propositionnelle, on utilise chaque affectation d'un paramètre comme un indicateur de la valeur de la fonction de skolem évaluée avec les paramètres correspondants. Dans notre exemple, on a la correspondance suivante :

$$s_1(x_1, x_2) = \begin{cases} (x_1 \wedge x_2) & \rightarrow y^1_{s_1} \\ (\neg x_1 \wedge x_2) & \rightarrow y^2_{s_1} \\ (x_1 \wedge \neg x_2) & \rightarrow y^3_{s_1} \\ (\neg x_1 \wedge \neg x_2) & \rightarrow y^4_{s_1} \end{cases}$$

$$s_2(x_1, x_2) = \begin{cases} (x_1) & \rightarrow y^1_{s_2} \\ (\neg x_1) & \rightarrow y^2_{s_2} \end{cases}$$

Pour $s_1(x_1, x_2)$ toutes les prémisses dans chaque implication sont fausses excepté pour $(x_1 \wedge \neg x_2 \rightarrow y^3_{s_1})$. Par conséquent, on capture le fait que la valeur de $s_1(x_1, x_2)$ correspond à celle de $y^3_{s_1}$. Bien évidemment, ce codage propositionnel n'est pas sous forme clausale et par conséquent, il doit être transformé en un ensemble de clauses :

$$s_1(x_1, x_2) = \begin{cases} (\neg x_1 \vee \neg x_2 \vee y^1_{s_1}) & \wedge \\ (x_1 \vee \neg x_2 \vee y^2_{s_1}) & \wedge \\ (\neg x_1 \vee x_2 \vee y^3_{s_1}) & \wedge \\ (x_1 \vee x_2 \vee y^4_{s_1}) \end{cases}$$

$$s_2(x_1, x_2) = \begin{cases} (\neg x_1 \vee y^1_{s_2}) & \wedge \\ (x_1 \vee y^2_{s_2}) \end{cases}$$

Maintenant, on peut substituer chaque fonction de skolem par sa forme propositionnelle correspondante. Ici on montre la substitution de s_2 dans la clause $(\neg x_1 \vee s_2(x_1))$:

$$\neg x_1 \vee ((\neg x_1 \vee y^1_{s_2}) \wedge (x_1 \vee y^2_{s_2}))$$

Cette substitution doit être convertie sous forme CNF comme montré dans la formule suivante :

$$(\neg x_1 \vee \neg x_1 \vee y^1_{s_2}) \wedge (\neg x_1 \vee x_1 \vee y^2_{s_2}) \leftrightarrow (\neg x_1 \vee y^1_{s_2})$$

Il est important de noter que cette conversion sous forme CNF ne doit pas faire intervenir des variables additionnelles. Sinon si l'on ajoute des variables existentielles supplémentaires à la formule, elles doivent être ajoutées dans le quantificateur existentiel le plus interne. Par conséquent, les variables universelles ne peuvent pas être éliminées par réduction-universelle et donc la formule finale restera toujours une formule QBF ce qui n'est pas le but escompté derrière cette transformation. Le principal problème à présent est la conversion qui peut faire croître considérablement la taille de la formule SAT résultante.

Néanmoins, sous forme clausale, des simplifications peuvent considérablement réduire la taille de la formule. (e.g., l'application de la réduction universelle des clauses et l'élimination des tautologies). Dans notre exemple on obtient seulement $y^1_{s_1}$ après ces deux opérations. La formule résultante ne contient aucune variable universelle dès lors que l'on applique la réduction universelle (toutes les variables existentielles de la formule précèdent les variables universelles). Par conséquent, la formule simplifiée est une formule CNF qui contient seulement des variables quantifiées existentiellement. On utilise ensuite un solveur SAT moderne (e.g., [MOSKEWICZ *et al.* 2001]) pour résoudre la formule sous forme CNF correspondante.

SKIZZO est basé sur l'approche présentée mais utilise d'autres techniques pour la conversion vers CNF et pour simplifier la formule propositionnelle résultante. Certaines de ces techniques sont basées sur le raisonnement symbolique. Malgré ces simplifications, la conversion d'une fonction skolemisée $\mathcal{F}$ en une formule booléenne CNF engendre une augmentation considérable de la taille de formule.

Les résultats expérimentaux de SKIZZO montrent que cette approche est compétitive sur un large panel de benchmarks [NARIZZANO *et al.* 2006]. Notons que ce solveur particulier propose un mélange de différentes techniques (e.g., recherche, raisonnement symbolique, résolution SAT) pour résoudre un problème QBF.

6.5.5 QBF et codage en BDD

Dans [AUDEMARD & SAIS 2005], les auteurs proposent une méthode pour résoudre les formules booléennes quantifiées en utilisant les $\mathcal{BDD}$. En effet, considérons la matrice $\mathcal{F}$. Les auteurs proposent de se passer du préfixe. Une recherche des modèles de $\mathcal{F}$ est effectuée sans tenir compte du préfixe, cette relaxation sur l'ordre des variables permet au solveur SAT utilisé d'effectuer une simple recherche de modèles et donc d'être plus efficace. A chaque fois qu'un modèle est trouvé, un impliquant premier est

extrait et codé dans le $\mathcal{BDD}$ et la recherche d'autres modèles est effectuée. Des opérations de simplifications sont effectuées dynamiquement sur le $\mathcal{BDD}$ en cours de résolution permettant ainsi de répondre à la question de la validité de la QBF.

7

Symétries et QBFs

Sommaire

7.1 Introduction 95
7.1.1 Symétries et SAT 96
7.1.2 Symétries et QBF 98
7.2 Détection de symétries : de SAT à QBF 99
7.3 Suppression des symétries dans les QBF 100
7.4 Approche basée sur les modèles (M-QSB) 102
7.4.1 M-QSB : Transformation générale 102
7.4.2 M-QSB : Transformation clausale 105
7.5 Approche basée sur les nogoods (N-QSB) 108
7.5.1 N-QSB : Élimination d'une seule symétrie 109
7.5.2 N-QSB : Élimination d'un ensemble de symétries 112
7.5.3 N-QSB : Transformation du préfixe 114
7.5.4 Out-littéraux positifs et négatifs 114
7.5.5 Complexité 115
7.6 Expérimentations 115
7.6.1 Résumé des résultats 116
7.6.2 Comparaison des résultats 117
7.7 Conclusion 117
7.8 Travaux futurs 122

Dans ce chapitre, nous proposons de supprimer les symétries dans les formules QBF. Si dans le cas SAT, le SBP (symmetry breaking predicates) est le plus utilisé pour supprimer les symétries. Dans le cas QBF, nous proposons deux approches différentes pour les supprimer.

La première approche [AUDEMARD *et al.* 2007b] constitue une extension du SBP de SAT vers QBF. Nous montrons dans ce cadre le fondement de cette extension qui consiste en plus du SBP classique à ajouter de nouvelles contraintes appelée QSBP (quantified symmetry breaking predicates) et un nouvel ordre des variables universelles qui interviennent dans les symétries est calculé via un graphe appelé graphe de précédence.

La deuxième approche [AUDEMARD *et al.* 2007a] proposée dans ce chapitre peut être vue comme une approche duale de la première. En effet, si dans le cadre SAT les interprétations symétriques ne peuvent être considérées que comme des contre-modèles de la formule, dans le cas QBF, et grâce à la présence des variables universelles, nous verrons que, sous certaines conditions, nous pouvons considérer certaines interprétations symétriques comme des modèles de la formule. Dans ce cadre, la formule est

transformée en une formule CNF/DNF. La forme CNF de la formule est constituée de la matrice de la QBF originale augmentée par le SBP classique projeté sur les parties existentielles des symétries. La partie DNF est un ensemble de modèles prédéfinis. Enfin une transformation de la CNF/DNF vers CNF est proposée. Des résultats expérimentaux sur un large panel d'instances symétriques montre l'intérêt de supprimer les symétries dans le cas QBF.

7.1 Introduction

Résoudre les formules booléennes quantifiées est devenu un domaine de recherche attractif et important durant les dernières années. Cet intérêt grandissant est dû au fait que plusieurs problèmes issus de l'intelligence artificielle (AI) (planification, raisonnement non monotone, vérification formelle etc.) peuvent être réduits aux QBFs, ce qui est considéré comme le problème canonique de la classe de complexité PSPACE. Une autre raison expliquant cet intérêt réside dans les progrès impressionnants réalisés dans la résolution du problème SAT. Plusieurs solveurs ont été proposés récemment pour la résolution du problème QBF (e.g. [GIUNCHIGLIA *et al.* 2006, ZHANG & MALIK 2002a, LETZ 2002a, BENEDETTI 2005a, SAMULOWITZ & BACCHUS 2005]). La majorité sont des extensions des solveurs SAT. Ceci n'est pas surprenant, du moment que le problème QBF est une extension naturelle du problème SAT, où les variables sont quantifiées avec les deux quantificateurs classiques $\exists$ et $\forall$. Une formule CNF classique est une QCNF où toutes les variables sont quantifiées existentiellement. La présence de quantificateurs universels accroît la difficulté du problème de décision (passant de NP-Complet à PSPACE-Complet). Cependant, un tel accroissement en terme de complexité dans le pire des cas rend le langage QBF plus expressif et plus approprié pour l'encodage *succinct* de quelques problèmes issus du monde réel. Une leçon à retenir à partir des progrès de la résolution du problème SAT est que le complexité dans le pire des cas n'est pas un obstacle pour résoudre efficacement de grandes instances issues des application liées au monde réel. En effet, plusieurs de ces problèmes contiennent des propriétés structurelles qui peuvent être bénéfiques pour l'étape de résolution. Les symétries constituent l'une de ces particularités structurelles. Elles sont reconnues comme très importantes pour de nombreux problèmes combinatoires intraitable (par exemple, Problème de Satisfaction de Contraintes (CSP), SAT et coloriage de graphes) et peuvent être d'un grand intérêt dans le contexte QBF.

Le but de ce travail est d'étendre les approches d'élimination de symétries de SAT au cas QBF. L'élimination des symétries est largement étudiée dans le contexte du problème de satisfaction de contraintes. L'ajout d'un ensemble de contraintes, appelé SBP ("symmetry breaking predicates"), est l'approche la plus utilisée pour supprimer les symétries. Les symétries sont éliminées en ajoutant de nouvelles contraintes (clauses dans le cas CNF) à la formule originale comme une étape de pré-traitement. L'un des avantages majeurs de cette approche est qu'elle est indépendante des solveurs considérés. Dans le cas QBF, la présence des quantificateurs universels rend cette extension plus complexe à réaliser. En effet, le SBP peut contenir des clauses ne contenant que des littéraux quantifiés universellement. Ajouter conjonctivement ce SBP à la formule QBF peut mener vers une formule non valide alors que la formule originale est valide. Un premier essai pour étendre les SBP aux QBFs est présenté dans [AUDEMARD *et al.* 2004]. Les auteurs proposent une approche QBF-SAT hybride où l'ensemble SBP est séparé de la formule QBF. Cependant cette approche dépend du solveur considéré et demande à être adaptée pour chaque solveur QBF.

Nous proposons dans la suite deux approches pour éliminer les symétries dans les QBFs. Les deux approches partagent la même idée à savoir la réécriture de la formule QBF en une formule asymétrique qui est équivalente pour la validité à l'originale. Dans la première méthode (basée sur les modèles), et contrairement au cas SAT où les interprétations symétriques ne peuvent être considérées comme modèles, nous montrons comment on peut transformer la QBF en y ajoutant des modèles prédéfinis ainsi qu'un

SBP limité aux variables existentielles de l'ensemble des symétries. Dans la seconde (basée sur les no-goods), on présente comment à l'aide de la transformation du préfixe et d'ajout de nouvelles contraintes QSBP en plus du SBP classique, on pourra éliminer les symétries dans le cas QBF.

Une validation expérimentale de ces deux approches est présentée dans la section 7.4. on montre que, sur de nombreux problèmes, nos approches permettent d'obtenir des améliorations significatives. Finalement quelques pistes prometteuses sont discutées après la conclusion.

7.1.1 Symétries et SAT

Avant de présenter notre cadre de suppression de symétries, il est nécessaire de rappeler quelques notions sur les symétries. Bien sûr, les symétries ont été largement étudiées dans plusieurs domaines, SAT [CRAWFORD 1992, BENHAMOU & SAIS 1994, ALOUL *et al.* 2003b], CSP [COHEN *et al.* 2005, ZHENG 2005], planification [FOX & LONG 1999] et coloriage de graphes [RAMANI *et al.* 2006].

Tout d'abord, introduisons quelques définitions de la théorie des groupes. Un groupe $(\mathcal{G}, \circ)$ est un ensemble fini $\mathcal{G}$ muni de l'opérateur binaire associatif $\circ : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$ admettant un élément neutre et un inverse. Evidemment, l'ensemble des permutations $\mathcal{P}$ sur un ensemble E associé à l'opérateur de composition $\circ$, noté $(\mathcal{P}, \circ)$, forme un groupe.

En outre, chaque permutation σ peut être représentée par un ensemble de cycles $(c_1 \dots c_n)$ où chaque cycle c_i est une liste d'éléments de E $(l_{i_1} \dots l_{i_{n_i}})$ telle que $\forall 1 \leq k < n_i, \sigma_i(l_{i_k}) = l_{i_{k+1}}$ et $\sigma_i(l_{i_{n_i}}) = l_{i_1}$. Nous définissons $|\sigma| = \sum_{c_i \in \sigma} |c_i|$ où $|c_i|$ est le nombre d'élèments de c_i. Dans cette partie, on ne considère que les symétries contenant des cycles binaires. En effet, dans la majorité des cas, les symétries peuvent être représentées par un ensemble de cycles binaires. Cependant, les deux approches proposées ici peuvent être étendues aux symétries avec des cycles de taille arbitraire.

Soit $\mathcal{F}$ une formule CNF, et σ une permutation sur $\mathcal{L}(\mathcal{F})$. Nous pouvons étendre la définition de la permutation σ à $\mathcal{F}$ comme suit : $\sigma(\mathcal{F}) = \{\sigma(c) | c \in \mathcal{F}\}$ et $\sigma(c) = \{\sigma(l) | l \in c\}$.

Deux types de symétries peuvent alors être définis en logique propositionnelle : les symétries sémantiques et les symétries syntaxiques. Étant donnée une formule $\mathcal{F}$, une symétrie sémantique est une permutation σ sur $\mathcal{L}(\mathcal{F})$ telle que $\sigma(\mathcal{F}) = \mathcal{F}$ où $\mathcal{F}$ et $\mathcal{F}$ sont équivalentes pour la logique propositionnelle i.e, $\mathcal{F} \models \sigma(\mathcal{F})$ et $\sigma(\mathcal{F}) \models \mathcal{F}$. Cette définition générale est difficile à exploiter en pratique. Il est en effet nécessaire de vérifier l'équivalence de deux formules. Les symétries syntaxiques, qui peuvent être considérées comme un cas spécial des symétries sémantiques peuvent être définies comme des permutation σ sur $\mathcal{L}(\mathcal{F})$ laissant $\mathcal{F}$ invariante i.e, $\sigma(\mathcal{F}) = \mathcal{F}$. La détection de telles symétries est équivalente au problème classique d'isomorphismes de graphes [CRAWFORD 1992], qui fait partie des problèmes ouverts dans la théorie de la complexité et qui n'est pas connu pour être NP-complet.

Aucun algorithme polynomial n'a été trouvé pour résoudre ce problème. Cependant, il existe des méthodes très efficaces pour détecter les symétries syntaxiques comme on peut le voir dans la section 7.2.

Les symétries syntaxiques ont été initialement définies par Krishnamurthy dans [KRISHNAMURTHY 1985]. La définition de la symétrie dans [KRISHNAMURTHY 1985] est restreinte aux variables de la symétrie (permutation sur un ensemble de variables) et a été étendue aux symétries de littéraux dans [BENHAMOU & SAIS 1994].

Définition 65 ([BENHAMOU & SAIS 1994]) *Soient $\mathcal{F}$ une formule CNF et σ une permutation des littéraux de $\mathcal{F}$, σ est une symétrie de $\mathcal{F}$ ssi elle satisfait les conditions suivantes :*

- $\sigma(\neg x) = \neg\sigma(x), \forall x \in \mathcal{L}(\mathcal{F})$
- $\sigma(\mathcal{F}) = \mathcal{F}$

À partir de la définition précédente, une symétrie σ définit une relation d'équivalence sur l'ensemble

des interprétations possibles. Pour tester la satisfiabilité, on a besoin uniquement de tester une interprétation par classe d'équivalence.

Proposition 2 *Soit $\mathcal{F}$ une formule CNF et σ une symétrie de $\mathcal{F}$. ρ est un modèle de $\mathcal{F}$ si et seulement si $\sigma(\rho)$ est aussi un modèle de $\mathcal{F}$.*

Plus intéressant encore, à partir de la proposition 2 on peut partitionner l'ensemble des modèles et l'ensemble des nogoods en classes d'équivalences. Ainsi, si une interprétation ρ n'est pas un modèle (nogood), on peut déduire d'autres nogoods (e.g. $\sigma(\rho)$). Éliminer les symétries consiste donc à éliminer toutes les interprétations symétriques sauf une. L'approche la plus utilisée pour éliminer les symétries consiste à ajouter des contraintes (clauses) - appelées "symmetry breaking predicates" ou "lex leader constraints" - à la formule originale [CRAWFORD 1992, CRAWFORD *et al.* 1996, WALSH 2006]. D'autres techniques de suppression de symétries ont été proposées. Par exemple, dans [BENHAMOU & SAIS 1994, BENHAMOU & SAÏDI 2007], les symétries sont détectées et exploitées dynamiquement. À chaque nœud de l'arbre de recherche si l'affectation d'un littéral x à $vrai$ mène vers un conflit, x et ses littéraux symétriques sont affectés simultanément à $faux$. Contrairement à l'approche SBP qui n'élimine que les symétries de la formule initiale (également appelées symétries globales), cette dernière approche détecte et exploite d'autres types de symétries (appelées symétries locales). Les symétries locales sont également référencées dans la littérature comme symétries conditionnelles [GENT *et al.* 2005].

Comme mentionné précédemment, nos travaux se limitent aux symétries globales et à leur suppression. Les deux approches qu'on va présenter peuvent être utilisées comme une technique de simplification (pré-traitement) et sont donc indépendantes du solveur utilisé ensuite pour la résolution.

Avant d'introduire la définition générale du SBP, illustrons l'idée derrière cette technique sur un exemple. Soit $\sigma = (x_1, y_1)$ une symétrie d'une formule CNF $\mathcal{F}$ qui contient seulement un seul cycle. Supposons que $\mathcal{F}$ admet $\rho = \{x_1, \neg y_1, \ldots\}$ comme modèle, donc $\sigma(\rho) = \{\neg x_1, y_1, \ldots\}$ est aussi un modèle de $\mathcal{F}$. Pour éliminer la symétrie ρ, il suffit d'imposer un ordre entre x_1 et y_1 en ajoutant la contrainte $x_1 \leq y_1$ qui peut être exprimée par la clause $c = (\neg x_1 \vee y_1)$ à la formule $\mathcal{F}$. Nous obtenons une nouvelle formule $\mathcal{F} = \mathcal{F} \cup c$ qui préserve l'équivalence pour la validité. Le modèle ρ de $\mathcal{F}$ est éliminé et considéré maintenant comme un contre-modèle de $\mathcal{F}$. Tous les autres modèles de $\mathcal{F}$ qui ne satisfont pas la clause binaire ajoutée sont aussi éliminés. Cette idée est généralisée dans la définition 66 pour une symétrie contenant un ensemble arbitraire de cycles.

Définition 66 *Soient $\mathcal{F}$ une CNF et $\sigma = (x_1, y_1)(x_2, y_2)\ldots(x_n, y_n)$ une symétrie de $\mathcal{F}$. Alors le "symmetry breaking predicates" (SBP), noté $sbp(\sigma)$, associé à σ est défini comme une conjonction de contraintes de la façon suivante :*

$$sbp(\sigma) = \begin{cases} (x_1 \leq y_1) \\ (x_1 = y_1) \rightarrow (x_2 \leq y_2) \\ (x_1 = y_1) \rightarrow (x_2 \leq y_2) \rightarrow (x_3 \leq y_3) \\ . \\ . \\ (x_1 = y_1) \wedge (x_2 = y_2)\ldots(x_{n-1} = y_{n-1}) \rightarrow (x_n \leq y_n) \end{cases}$$

La propriété suivante montre que l'approche SBP préserve l'équivalence pour la validité entre la formule originale et celle générée.

Proposition 3 *Soit $\mathcal{F}$ une formule booléenne et σ une symétrie de $\mathcal{F}$. Alors $\mathcal{F}$ et $(\mathcal{F} \wedge sbp(\sigma))$ sont équivalentes pour la validité.*

Dans le but de limiter l'explosion combinatoire de la transformation clausale des prédicats, une variable supplémentaire α_i est ajoutée par cycle (x_i, y_i) pour exprimer l'égalité entre x_i et le y_i. Cependant, l'un des inconvénients majeurs de cette approche est la taille du SBP qui est exponentiel dans le pire des cas. Récemment, des réductions intéressantes de la taille du SBP ont été obtenues dans [ALOUL *et al.* 2006].

7.1.2 Symétries et QBF

Nous pouvons maintenant définir les symétries dans le cadre QBF. Définissons tout d'abord l'extension de permutations dans le cas QBF. Les permutations doivent prendre en compte le préfixe de la QBF. Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \ldots, \mathcal{Q}_n X_n \mathcal{M}$ une QBF et σ une permutation de $\mathcal{L}(\mathcal{F})$, alors
$\sigma(\mathcal{F}) = \mathcal{Q}_1 \sigma(X_1) \ldots \mathcal{Q}_n \sigma(X_n) \sigma(\mathcal{M})$. La définition 65 peut être étendue au cas QBF comme suit :

Définition 67 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1, \ldots, \mathcal{Q}_n X_n \mathcal{M}$ une formule booléenne quantifiée. σ est une permutation des littéraux de $\mathcal{F}$. σ est une symétrie de $\mathcal{F}$ ssi*
- *$\sigma(\neg x) = \neg\sigma(x), \forall x \in \mathcal{L}(\mathcal{F})$*
- *$\sigma(\mathcal{F}) = \mathcal{F}$ i.e, $\sigma(\mathcal{M}) = \mathcal{M}$ et $\forall i \in \{1, \ldots, n\}\ \sigma(X_i) = X_i$.*

La définition précédente restreint les permutations entre les variables du même groupe de quantificateurs. Par conséquent, chaque symétrie σ d'une QBF $\mathcal{F}$ est aussi une symétrie de la formule booléenne $\mathcal{M}$. L'inverse n'est pas vrai. L'ensemble des symétries de $\mathcal{F}$ est un sous-ensemble des symétries de $\mathcal{M}$.

Comme les variables, dans le cas QBF, peuvent être quantifiées soit existentiellement ou universellement, on distingue trois types de symétries. La définition suivante fait l'inventaire de ces trois types de symétries.

Définition 68 *Soit $\mathcal{F}$ une QBF, σ une symétrie de $\mathcal{F}$ et $c = (x, y)$ un cycle de σ. Un cycle c est dit universel (resp. existentiel) ssi x et y sont universellement quantifiés (resp. existentiellement quantifiés). Une symétrie σ est dite universelle (resp. totalement universelle) ssi elle contient au moins un cycle universel (resp tous les cycles sont quantifiés universellement). Autrement, elle est dite existentielle.*

Comme pour SAT (voir proposition 2), la proposition suivante exhibe une propriété importante qui permet de réduire l'espace de recherche.

Proposition 4 *Soit σ une symétrie d'une QBF $\mathcal{F} = \mathcal{Q}_1 X_1, \ldots, \mathcal{Q}_n X_n \mathcal{M}$, π est un ensemble d'interprétations sur $\mathcal{V}(\mathcal{F})$. π est une politique totale de $\mathcal{F}$, ssi $\sigma(\pi)$ est une politique totale de $\mathcal{F}$.*

Preuve :
Pour la preuve de la propriété précédente, on réécrit le préfixe de $\mathcal{F}$ sous la forme :
$\mathcal{F} = \mathcal{Q}_n X_n, \ldots, \mathcal{Q}_1 X_1 \mathcal{M}$.

Par induction sur n, on montre que si π est une politique totale alors $\sigma(\pi)$ est également une politique totale de $\mathcal{F}$. Pour $n = 1$, on ne considère que le cas où $\mathcal{Q}_1 = \exists$ (le quantificateur le plus interne est existentiel). Dans ce cas toutes les variables de $\mathcal{F}$ sont existentiellement quantifiées et $\mathcal{F}$ peut être considérée comme une simple formule booléenne. $\sigma(\pi)$ est donc aussi un ensemble de modèles (cas propositionnel des symétries). Maintenant supposons que la proposition soit vraie pour $m-1$, montrons qu'elle est vraie pour n, c'est-à-dire π est une politique totale pour $\mathcal{F} = \mathcal{Q}_1 X_1, \ldots, \mathcal{Q}_n X_n \mathcal{M}$. Nous devons considérer deux cas :

1. $\mathcal{Q}_n = \forall$: par définition d'une politique totale $\forall \overrightarrow{x}_n \in 2^{X_n}\ (\pi \backslash \overrightarrow{x}_n)$ est également une politique totale $Q_{n-1} X_{n-1}, \ldots, \mathcal{Q}_1 X_1 \mathcal{M}(\overrightarrow{x}_n)$. Par hypothèse, $\sigma(\pi \backslash \overrightarrow{x}_n)$ est une politique totale de $Q_{n-1} X_{n-1}, \ldots, \mathcal{Q}_1 X_1 \mathcal{M}(\overrightarrow{x}_n)$. $\forall \overrightarrow{x}_n \in 2^{X_n}$. Nous avons aussi $\sigma(\overrightarrow{x}_n) \in 2^{X_n}$. Par conséquent $\bigcup_{\overrightarrow{x}_n} \{\sigma(\overrightarrow{x}_n).\sigma(\pi \backslash \overrightarrow{x}_n)\} = \sigma(\pi)$ est une politique totale de $\mathcal{F}$.

2. $\mathcal{Q}_n = \exists : (\pi[X_n]) = \{\overrightarrow{x}_n\}$ et $\pi \backslash \overrightarrow{x}_n$ est une politique totale de $Q_{n-1}X_{n-1}, \ldots, \mathcal{Q}_1 X_1 \mathcal{M}(\overrightarrow{x}_n)$. En utilisant l'hypothèse d'induction $\sigma(\pi \backslash \overrightarrow{x}_n)$ est aussi une politique totale.
Donc $\sigma(\pi) = \sigma(\overrightarrow{x}_n).\sigma(\pi \backslash \overrightarrow{x}_n)$ est une politique totale $\mathcal{F}$.

L'implication dans l'autre sens peut être prouvée de la même façon en utilisant σ^{-1}. □

Exemple 20 *Soit la QBF suivante*

$$\mathcal{F} = \forall x_1 x_2 \exists x_3 x_4 \left\{ \begin{array}{lcl} (\neg x_1 \vee \neg x_2 \vee x_3) & \wedge & (\neg x_1 \vee \neg x_2 \vee x_4) \quad \wedge \\ (x_1 \vee \neg x_3 \vee \neg x_4) & \wedge & (x_2 \vee \neg x_3 \vee \neg x_4) \quad \wedge \end{array} \right.$$

Les permutations $\sigma_1 = (x_1, x_2)(x_3)(x_4)$ *et* $\sigma_2 = (x_1)(x_2)(x_3, x_4)$ *sont des symétries de* $\mathcal{F}$*. Dans la suite de l'exemple, nous supprimons les littéraux qui ne permutent qu'avec eux-mêmes (comme* x_3 *et* x_4 *dans* σ_1*). La formule* $\mathcal{F}$ *est satisfiable.*
i.e, $\pi = \{(\neg x_1, \neg x_2, \neg x_3, \neg x_4), (\neg x_1, x_2, \neg x_3, x_4), (x_1, \neg x_2, x_3, \neg x_4), (x_1, x_2, x_3, x_4)\}$ *est une politique totale de* $\mathcal{F}$*. Nous pouvons vérifier que* $\sigma_1(\pi)$ *et* $\sigma_2(\pi)$ *sont aussi des politiques totales de* $\mathcal{F}$*.*

$\sigma_1(\pi) = \{(\neg x_2, \neg x_1, \neg x_3, \neg x_4), (\neg x_2, x_1, \neg x_3, x_4), (x_2, \neg x_1, x_3, \neg x_4), (x_2, x_1, x_3, x_4)\}$
$\sigma_2(\pi) = \{(\neg x_1, \neg x_2, \neg x_4, \neg x_3), (\neg x_1, x_2, \neg x_4, x_3), (x_1, \neg x_2, x_4, \neg x_3), (x_1, x_2, x_4, x_3)\}$

Dans le contexte des formules booléennes, les symétries peuvent être considérées comme une relation d'équivalence sur l'ensemble des interprétations de la formule booléenne, induisant ainsi des classes d'équivalence, qui contiennent des modèles ou nogoods. La notion de symétrie dans les QBF étend cette notion d'équivalence à des ensembles d'interprétations. Chaque classe d'équivalence contient soit des politiques totales, soit le contraire.

7.2 Détection de symétries : de SAT à QBF

Différentes techniques pour détecter les symétries dans les formules booléennes ont été proposées. Quelques unes d'entre elles traitent directement la formule booléenne [BENHAMOU & SAIS 1994].

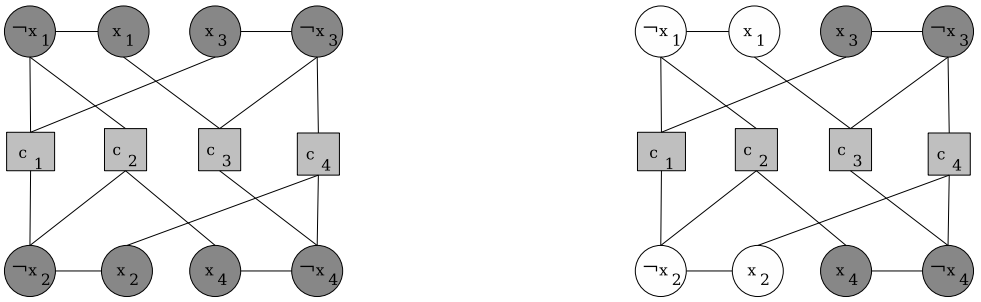
Une autre technique communément utilisée consiste à réduire le problème de détection de symétries à celui de recherche d'isomorphismes de graphes (i.e, problème de déterminer s'il existe une bijection entre deux graphes donnés) [CRAWFORD 1992, CRAWFORD *et al.* 1996]. Le problème d'isomorphisme de graphe est un problème ouvert dans la théorie de la complexité. Dans notre contexte, nous traitons principalement du problème d'automorphisme de graphe (i.e, trouver une bijection qui lie un graphe à lui-même) qui est un cas particulier d'isomorphisme de graphes. De nombreux algorithmes ont été proposés pour calculer les automorphismes de graphes. Nous mentionnons NAUTY [MCKAY 1990] un des plus efficaces.

Récemment, Aloul et al. [ALOUL *et al.* 2003b] ont proposé une technique intéressante qui transforme la formule CNF $\mathcal{F}$ en un graphe $G_{\mathcal{F}}$ dans lequel les sommets sont coloriés. Cette coloration est importante lors de la recherche d'automorphismes, puisque deux sommets de couleurs différentes ne pourront pas être permutés entre eux. La translation est faite de la manière suivante et est illustrée dans l'exemple 21.

- Chaque variable est représentée par deux sommets, un pour le littéral positif et l'autre pour son opposé. Une couleur gris foncé est associée à ces sommets.
- Une arête relie deux littéraux opposés.
- Chaque clause non binaire est représentée par un sommet de couleur gris clair, et une arête relie cette clause à tous les littéraux qui la composent.
- Chaque clause binaire $l_1 \vee l_2$ est représentée par une arête double entre les sommets l_1 et l_2.

Exemple 21 *Soit $\mathcal{M}$ la formule CNF de l'exemple 20.a. La figure 7.1 représente le graphe $G_{\mathcal{M}} = (V, E)$ où V contient $2 \times |\mathcal{L}(\mathcal{M})| = 8$ sommets (couleur gris foncé) associés à $\mathcal{L}(\mathcal{M})$ et 4 sommets (couleur gris clair) correspondant aux clauses de $\mathcal{M}$. L'ensemble des arrêtes E relie les littéraux à leurs opposés (e.g. $(x_1, \neg x_1)$ et les clauses aux littéraux qui la composent (i.e, $(c_1, \neg x_1), (c_1, \neg x_2)$ et (c_1, x_3)). Trois automorphismes laissent invariant ce graphe $G_{\mathcal{M}}$ $a_1 = (x_1, x_2)(x_3)(x_4)\ [(c_1)(c_2)(c_3, c_4)]$, $a_2 = (x_1)(x_2)(x_3, x_4)[(c_1, c_2)(c_3)(c_4)]$, et $a_3 = (x_1, x_3)(x_2, x_4)[(c_1, c_3)(c_2, c_4)]$. Ceux-ci correspondent aux symétries σ_1, σ_2 et σ_3, obtenues par projection de a_1, a_2 and a_3 sur les littéraux.*

Il n'est pas difficile d'étendre la transformation précédente aux formules QBF. Toutefois, il faut prendre en compte le préfixe de la formule QBF (voir la second point de la définition 67) et donc empêcher les permutations entre les littéraux de groupes de quantificateurs distincts. Pour ce faire, nous ajoutons une couleur par groupe de quantificateurs et imposons la même couleur aux sommets du même groupe. Ensuite, nous recherchons les automorphismes de graphes (à l'aide de Nauty par exemple) sur le graphe résultant. L'exemple 22 illustre cette transformation.



a. De CNF au graphe b. De QBF au graphe

FIG. 7.1 – Réduction du graphe

Exemple 22 *Soit $\mathcal{F} = \forall x_1 x_2 \exists x_3 x_4 \mathcal{M}$ la formule QBF de l'exemple 20. La figure 7.1.b montre son graphe associé $G_{\mathcal{F}}$. Ici, on a 3 couleurs, une pour les clauses non binaires (gris clair), une pour le premier groupe de quantificateurs universel (blanc), et la dernière pour le groupe existentiel (gris foncé). Cette formule QBF $\mathcal{F}$ a deux symétries (x_1, x_2) et (x_3, x_4). La symétrie $(x_1, x_3)(x_2, x_4)$ de $\mathcal{M}$ (voir exemple 21) n'est pas une symétrie $\mathcal{F}$, x_1 et x_3 n'appartiennent pas au même groupe de quantificateurs.*

Les deux prochaines sections détaillent les deux méthodes que nous introduisons pour supprimer les symétries dans les QBF.

7.3 Suppression des symétries dans les QBF

Pour générer le SBP, il faut choisir un ordre des variables afin d'ordonner les cycles. Par conséquent, le SBP correspondant dépend de l'ordre imposé. Dans le cas SAT, les générateurs classiques utilisent l'ordre lexicographique défini comme suit :

Définition 69 (ordre lexicographique) *Soit $\sigma = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ un ensemble de cycles. σ est ordonnée lexicographiquement (lex-ordonné en plus court) ssi $x_i > 0$, $x_i \leq |y_i|$ $1 \leq i \leq n$ et $x_i < x_{i+1}$ $1 \leq i < n$*

Dans le cas QBF, un tel ordre doit respecter l'ordre partiel imposé par le préfixe. En effet, les cycles doivent être ordonnés suivant l'ordre du préfixe, et pour les cycles appartenant au même groupe de quantificateurs on dispose d'une liberté pour les ordonner.

Définition 70 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_i X_i \dots \mathcal{Q}_n X_n \, \mathcal{M}$ une QBF et σ une symétrie de $\mathcal{F}$. Nous définissons $\sigma[X_i]$ comme $\{\sigma_j | \sigma_j = (x_j, y_j) \in \sigma, |x_j| \in X_i, |y_j| \in X_i\}$. En respectant l'ordre du préfixe, la symétrie σ peut être réécrite comme $\sigma' = \{\sigma'_1 \dots \sigma'_i \dots \sigma'_m\}$ tel que $\sigma'_i = \sigma[X_i]$. σ' est appelée p-ordonnée.*

Exemple 23 *Soit*

$$\mathcal{F} = \exists x_5 x_6 \forall x_1 x_2 \exists x_3 x_4 \left\{ \begin{array}{lllll} (\neg x_1 \vee x_2 \vee x_3) & \wedge & (x_1 \vee \neg x_2 \vee x_4) & \wedge \\ (x_3 \vee x_5) & \wedge & (x_4 \vee x_6) & \wedge \\ (x_1 \vee \neg x_5) & \wedge & (x_2 \vee \neg x_6) \end{array} \right.$$

$\mathcal{F}$ admet une symétrie σ. La génération de cette symétrie avec l'ordre lexicographique donne $\sigma = \{(x_1, x_2)(x_3, x_4)(x_5, x_6)\}$. Réordonnée σ en respectant le préfixe de $\mathcal{F}$ cela donne $\sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ avec $\sigma_1 = \sigma[X_1] = (x_5, x_6)$, $\sigma_2 = \sigma[X_2] = (x_1, x_2)$, $\sigma_3 = \sigma[X_3] = (x_3, x_4)$. La forme p-ordonnée de σ est $\sigma = \{\sigma_1, \sigma_2, \sigma_3\} = \{(x_5, x_6)(x_1, x_2)(x_3, x_4)\}$

Dans la suite du chapitre les symétries sont considérées p-ordonnées.

Avant de proposer nos deux méthodes pour supprimer les symétries dans les QBF, nous souhaitons traiter le cas des symétries existentielles et monter aussi pourquoi le suppression des symétries universelles par ajout de contraintes (SBP) est un problème non trivial.

Pour les symétries existentielles, le SBP classique [CRAWFORD *et al.* 1996] peut être transformé linéairement sous forme CNF grâce a des variables supplémentaires quantifiées existentiellement. L'ensemble obtenu peut être ajouté conjonctivement à la formule originale en préservant sa validité. Ceci est formalisé dans la proposition 5.

Proposition 5 *Soient $\mathcal{F} = QX\mathcal{M}$ une formule booléenne quantifiée et σ une symétrie existentielle de $\mathcal{F}$. Alors $\mathcal{F}$ est valide si et seulement si $QX(\mathcal{M} \wedge sbp(\sigma))$ est valide.*

Preuve : Comme σ est une symétrie existentielle, le $sbp(\sigma)$ généré contient seulement des littéraux existentiels. Par conséquent, la preuve est similaire à celle du cas SAT [CRAWFORD *et al.* 1996]. □

Comme illustré dans l'exemple 24, ajouter conjonctivement le SBP classique dans le cas QBF ne permet pas de préserver l'équivalence pour la validité.

Exemple 24 *Soit la QBF suivante :*

$$\mathcal{F} = \forall x_1 y_1 \exists x_2 y_2 \left\{ \begin{array}{l} (x_1 \vee \neg x_2) \\ (y_1 \vee \neg y_2) \\ (\neg x_1 \vee \neg y_1 \vee x_2 \vee y_2) \end{array} \right.$$

La permutation $\sigma = \{(x_1, y_1)(x_2, y_2)\}$ est une symétrie de $\mathcal{F}$. Supprimer la symétrie σ en utilisant l'approche traditionnelle induit $sbp(\sigma)$ qui contient deux contraintes $(x_1 \leq y_1)$ et $(x_1 = y_1) \rightarrow (x_2 \leq y_2)$. Sous forme CNF le $sbp(\sigma)$ est l'ensemble de clauses $\{(\neg x_1 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_2) \wedge (y_1 \vee \neg x_2 \vee y_2)\}$. Comme la clause $(\neg x_1 \vee y_1)$ est totalement universelle, la nouvelle QBF obtenue $\mathcal{F}' = \forall x_1 y_1 \exists x_2 y_2 \, \mathcal{M} \wedge sbp(\sigma)$ est non valide alors que la formule originale est valide.

Les deux approches que nous proposons pour éliminer les symétries universelles répondent à la difficulté posée par l'exemple 24. Ces deux approches introduisent des variables supplémentaires dans

le but de réécrire la formule originale. Le résultat est donc une formule asymétrique équivalente à la formule originale pour la validité. Avant de décrire chacune de ces approches nous commençons par illustrer brièvement leurs similitudes et leurs différences.

Considérons tout d'abord l'exemple 24. Les deux interprétations partielles $\rho_1 = \{x_1, \neg y_1\}$ et $\rho_2 = \{\neg x_1, y_1\}$ sont symétriques. Supposons que la variable x_1 soit affectée avant y_1. Dans les deux approches l'une de ces interprétations doit être éliminée (par exemple ρ_1). Les deux approches simulent différentes éliminations. Plus précisément, la QBF et le SBP sont réécrits tels que pour la première approche (respectivement la deuxième) la formule $\mathcal{F}(\rho_1)$ est considérée satisfiable (respectivement insatisfiable). À partir de cette dualité et pour raison de simplicité, la première approche est appelée approche basée sur les modèles (M-QSB), alors que la seconde est appelée approche basée sur les nogoods (N-QSB).

7.4 Approche basée sur les modèles (M-QSB)

7.4.1 M-QSB : Transformation générale

Nous commençons notre raisonnement en ne prenant en compte que les symétries totalement universelles. Comme expliqué précédemment, pour une QBF donnée $\mathcal{F} = QX\mathcal{M}$ et une symétrie $\sigma = (x_1, y_1)$ totalement universelle avec un seul cycle, l'une des deux interprétations $\rho_1 = \{x_1, \neg y_1\}$ ou $\rho_2 = \{\neg x_1, y_1\}$ doit être éliminée. Dans l'approche *M-QSB*, et contrairement au cas existentiel, comme x_1, y_1 sont quantifiés universellement, on peut considérer l'une des deux interprétations comme modèle de $\mathcal{M}$. Cette propriété intéressante ne peut être utilisée dans le cas SAT où les interprétations symétriques non parcourues sont considérées comme étant en conflit avec la formule originale.

Donc au lieu de trouver un mécanisme pour impliquer y_1 à vrai lorsque x_1 est à $vrai$, $\neg sbp(\sigma)$ est ajoutée disjonctivement à $\mathcal{M}$ permettant d'obtenir ainsi une nouvelle formule $\mathcal{F}^m = QX\mathcal{M}'$ telle que $\mathcal{M}' = (\mathcal{M} \vee \neg sbp(\sigma))$. La formule $\mathcal{F}^m$ est équivalente à $\mathcal{F}$ pour la validité. En effet, comme $sbp(\sigma)(\rho_1)$ est à $faux$, la nouvelle matrice $\mathcal{M}'$ est trivialement satisfaite par ρ_1, i.e, ρ_1 est un modèle de $\mathcal{M}'$. Pour l'interprétation ρ_2, le $sbp(\sigma)(\rho_2)$ est à $vrai$ et les deux formules $\mathcal{M}'(\rho_2)$ et $\mathcal{M}(\rho_2)$ représentent la même formule.

La majorité des solveurs QBF modernes basés sur DPLL opère sur une formule QBF sous forme CNF/DNF. La formule CNF est la base de clauses augementée par les clauses apprises durant la recherche (formule DNF). La DNF est un ensemble de cubes obtenus à partir des analyses de solutions permettant d'éviter les modèles déjà trouvés. Dans ce cas, l'interprétation ρ_2 est ajoutée préalablement à la DNF. L'algorithme de recherche disposera donc d'un modèle prédéfini.

Dans la proposition suivante, on généralise cette idée pour toute symétrie totalement universelle de taille quelconque.

Proposition 6 *Soit $\mathcal{F} = QX\mathcal{M}$ une formule QBF et σ une symétrie totalement universelle de $\mathcal{F}$. $\mathcal{F} = \mathcal{Q}X\mathcal{M}$ est satisfiable si et seulement si $\mathcal{F}^m = \mathcal{Q}X\mathcal{M} \vee \neg sbp(\sigma)$ est satisfiable.*

Preuve :
$\rightarrow$) : Il est évident que chaque modèle de $\mathcal{M}$ est aussi un modèle de $\mathcal{M} \vee \neg sbp(\sigma)$. Par conséquent, une politique totale de $\mathcal{F}$ est aussi une politique totale de $QX(\mathcal{M} \vee \neg sbp(\sigma))$.

$\leftarrow$) : Soit π une politique de $QX(\mathcal{M} \vee \neg sbp(\sigma))$, on construit une politique π' pour $\mathcal{F}$. Soit $\rho \in \pi$, deux cas doivent être distingués :

- $\rho \models \mathcal{M}$, alors $\pi' = \pi' \cup \rho$
- $\rho \not\models \mathcal{M}$, donc, $\rho \models \neg sbp(\sigma)$, $\exists c = (x_i, y_i) \in \sigma$ où c est universel et ρ est de la forme $(x_1, \ldots, x_i, \neg y_i, \ldots)$. Comme π est une politique totale de $\mathcal{M} \vee \neg sbp(\sigma)$ alors $\exists \rho' \in \pi$ de la forme

$(x_1, \ldots, \neg x_i, y_i, \ldots)$ tel que $\rho' \models \mathcal{M} \vee \neg sbp(\sigma)$ et $\rho' \not\models \neg sbp(\sigma)$. Par conséquent, $\rho' \models \mathcal{M}$ et $\sigma(\rho') \models \mathcal{M}$ où $\sigma(\rho')$ et de la forme $(x_1, \ldots, x_i, \neg y_i, \ldots)$. Donc $\pi' = \pi' \cup \sigma(\rho')$.
Par construction, il est clair que π' est une politique totale de $\mathcal{F}$. □

Exemple 25 *Soit*

$$\mathcal{F} = \forall x_1 \exists x_2 y_2 \left\{ \begin{array}{lll} (x_1 \vee x_2 \vee \neg y_2) & \wedge & (x_1 \vee \neg x_2 \vee y_2) \quad \wedge \\ (\neg x_1 \vee \neg x_2 \vee y_2) & \wedge & (\neg x_1 \vee x_2 \vee \neg y_2) \end{array} \right.$$

La formule $\mathcal{F}$ admet la symétrie $\sigma = (x_1, \neg x_1)$. Comme $(x_1, \neg x_1)$ est un cycle universel, et $sbp(\sigma) = (\neg x_1)$, nous avons : $\mathcal{F}^m = \mathcal{F} \vee \neg sbp(\sigma) = \mathcal{F} \vee x_1$. En transformant $\mathcal{F}^m$ sous forme clausale on obtient : $\mathcal{F}^m = \forall x_1 \exists x_2 y_2 \; (x_1 \vee x_2 \vee \neg y_2) \wedge (x_1 \vee \neg x_2 \vee y_2)$

Le problème survient lorsque la symétrie σ contient à la fois des cycles universels et des cycles existentiels. Dans ce cas, la nouvelle formule $\mathcal{F}^m = QX(\mathcal{M} \vee \neg sbp(\sigma))$ décrite dans la proposition 6 n'est pas équivalente à $\mathcal{F}$ pour la validité. Considérons en effet un contre exemple. Soit $\sigma = (x_1, y_1) \ldots (x_n, y_n)$ une symétrie telle que $\sigma[X_1] = (x_1, y_1)$ et $\mathcal{Q}_1 = \exists$. Si on affecte x_1 à $vrai$ et y_1 à $faux$, alors le $sbp(\sigma)$ devient $faux$ et $(\mathcal{M} \vee \neg sbp(\sigma))$ est satisfiable. Nous concluons donc que toutes les formules $\mathcal{F}$ qui admettent une symétrie $\sigma = (x_1, y_1) \ldots (x_n, y_n)$ telle que $\sigma[X_1] = (x_1, y_1)$ et $\mathcal{Q}_1 = \exists$ sont satisfiables. Cette dernière affirmation est clairement fausse.

Résumons les deux propriétés prouvées jusqu'à présent. Pour une symétrie σ de $\mathcal{F} = QX\mathcal{M}$, si tous les cycles de σ sont existentiels (respectivement universels) alors $\mathcal{F}$ et $(QX(\mathcal{M} \wedge sbp(\sigma))$ (respectivement $(QX(\mathcal{M} \vee \neg sbp(\sigma)))$ sont équivalentes pour la validité.

Pour les symétries contenant à la fois des cycles universels et existentiels, la nouvelle QBF est obtenue en alternant les sous-parties des deux formules $(\mathcal{M} \vee \neg sbp(\sigma))$ et $(\mathcal{M} \wedge sbp(\sigma))$.

Définition 71 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \ldots \mathcal{Q}_n X_n \mathcal{M}$ une QBF et σ une symétrie de $\mathcal{F}$. Nous définissons $\sigma \downarrow X_i$ comme $\bigcup_{1 \leq j \leq i} \sigma[X_j]$. Aussi, $\sigma \uparrow X_i$ est définie par $\sigma \backslash (\sigma \downarrow X_i)$*

Il est important de noter que, étant donnée une symétrie σ totalement universelle (respectivement existentielle), de $\mathcal{F}$, $sbp_{\sigma \downarrow X_i} \subseteq sbp(\sigma)$. Dans la proposition suivante, on montre qu'étant donnée une symétrie existentielle (respectivement universelle) σ, on peut décider d'éliminer une partie de la symétrie $\sigma \downarrow X_i$ en préservant la satisfiabilité.

Proposition 7 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \ldots \mathcal{Q}_n X_n \mathcal{M}$ (ou encore $QX\mathcal{M}$) une QBF et σ une symétrie de $\mathcal{F}$. Si σ est existentielle (respectivement totalement universelle) alors $\mathcal{F}$ et $QX(\mathcal{M} \wedge sbp_{\sigma \downarrow X_i})$ (respectivement $QX(\mathcal{M} \vee \neg sbp_{\sigma \downarrow X_i})$ sont équivalentes pour la validité.*

Définition 72 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \ldots \mathcal{Q}_n X_n \mathcal{M}$ une QBF et σ une symétrie de $\mathcal{F}$. Nous définissons $sbp(\sigma)[X_i]$ comme suit :*

- $sbp(\sigma)[X_1] = sbp_{\sigma \downarrow X_1}$
- $sbp(\sigma)[X_i] = (sbp_{\sigma \downarrow X_i}) \backslash (sbp_{\sigma \downarrow X_{i-1}}), i > 1$

A partir de la définition ci-dessus, il est facile de voir que l'ensemble $\{sbp(\sigma)[X_1] \ldots sbp(\sigma)[X_n]\}$ est une partition de $sbp(\sigma)$.

Exemple 26 *Soient $\mathcal{F} = \forall X_1 \exists Y_1 \mathcal{M}$ une formule QBF telle que $X_1 = \{x_1, y_1, x_2, y_2\}$ et $Y_1 = \{x_3, y_3\}$ et $\sigma = \{(x_1, y_1)(x_2, y_2)(x_3, y_3)\}$ une symétrie de $\mathcal{F}$.*

$$\begin{aligned} sbp(\sigma)[X_1] = \quad & x_1 \leq y_1 \\ & (x_1 = y_1) \rightarrow x_2 \leq y_2 \\ sbp(\sigma)[X_2] = \quad & (x_1 = y_1) \wedge (x_2 = y_2) \rightarrow x_3 \leq y_3 \end{aligned}$$

En utilisant les définitions précédentes, on peut maintenant décrire la formulation de la suppression des symétries basée sur les modèles pour les QBF comme suit :

Proposition 8 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \dots Q_{n-1} X_{n-1} \exists X_n \mathcal{M}$ une QBF et $\sigma = \{\sigma_1 \dots \sigma_n\}$ une symétrie de $\mathcal{F}$ telle que $\forall 1 \leq i \leq n\ \sigma_i = \sigma[X_i]$. $\mathcal{F}^n = \mathcal{Q}_1 X_1 \dots \exists X_n (\dots (((\mathcal{M} \wedge sbp(\sigma)[X_n]) \vee \neg sbp(\sigma)[X_{n-1}]) \wedge sbp(\sigma)[X_{n-2}]) \vee \dots \circ \mathcal{M}')$ tel que $\circ = \wedge$ et $\mathcal{M}' = sbp(\sigma)[X1]$ (respectivement $\circ = \vee$ et $\mathcal{M}' = \neg sbp(\sigma)[X1]$) si $\mathcal{Q}_1 = \exists$ (respectivement $\mathcal{Q}_1 = \forall$). $\mathcal{F}$ et $\mathcal{F}^m$ sont équivalentes pour la validité.*

Preuve : Distinguons deux cas :

- $\mathcal{Q}_1 = \forall$: comme $\sigma[X_1]$ est totalement universelle, on déduit que $\mathcal{F}$ et $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n (\mathcal{M} \vee \neg sbp(\sigma)[X_1])$ sont équivalentes par rapport à la validité (voir proposition 6 et 7).
  Soit $\sigma[X_1] = (x_1, y_1) \dots (x_p, y_p)$ et $\eta_1 = (x_1 = y_1) \wedge \dots \wedge (x_p = y_p)$. Si $\eta_1 = vrai$ alors $\{\sigma_2 \dots \sigma_m\}$ est une symétrie de $Q_2 X_2 \dots \mathcal{Q}_n X_n \mathcal{M}$ et aussi une symétrie de $QX(\mathcal{M} \vee \neg sbp(\sigma)[X_1])$. En considérant $\sigma[X_2]$ avec $\eta_1 = vrai$, $QX\mathcal{M}$ est équivalente à
  $QX(\mathcal{M} \vee \neg sbp(\sigma)[X_1]) \wedge (\neg \eta_1 \vee sbp_{\sigma_2})$. A partir de la définition 72, On peut aisément conclure que $sbp(\sigma)[X_2] = (\neg \eta_1 \vee sbp_{\sigma_2})$. La preuve se construit ainsi en itérant le même raisonnement.
- $\mathcal{Q}_1 = \exists$: la preuve est obtenue avec le même raisonnement □

Exemple 27 *Considérons la formule $\mathcal{F}$ donnée dans l'exemple 23. $\mathcal{F}$ est réécrite sous forme : $\mathcal{F}^m = \exists x_1 y_1 \forall x_2 y_2 \exists x_3 y_3\ (((\mathcal{M} \wedge sbp(\sigma)[X_3]) \vee \neg sbp(\sigma)[X_2]) \wedge sbp(\sigma)[X_1])$ telle que $sbp(\sigma)[X_1] = x_1 \leq y_1$, $sbp(\sigma)[X_2] = (x_1 = y_1) \rightarrow x_2 \leq y_2$ et $sbp(\sigma)[X_3] = (x_1 = y_1) \wedge (x_2 = y_2) \rightarrow x_3 \leq y_3$*

La proposition 8 peut aisément être généralisée à un ensemble de symétries $S = \{\sigma^1, \dots, \sigma^k\}$ en considérant les projections de toutes les symétries de S sur X_i i.e, $S \downarrow X_i = \bigcup_{1 \leq j \leq k} \sigma^j \downarrow X_i$. Une telle généralisation peut être obtenue en remplaçant une symétrie σ par l'ensemble des symétries S dans la définition 72 et la proposition 8.

Soient $\mathcal{F} = QX\mathcal{M}$ une formule booléenne quantifiée et $\sigma = (x_1, y_1) \dots (x_n, y_n)$ une symétrie universelle de $\mathcal{F}$. Soit ρ une affectation de tous les littéraux apparaissant dans les $i-1$ premiers cycles de σ. Supposons que ρ satisfait $(x_1 = y_1) \wedge (x_2 = y_2) \dots \wedge (x_{i-1} = y_{i-1})$, l'approche proposée distingue les deux cas suivants :

1. $q(x) = \exists$: Si $x_i = vrai$ alors y_i doit être affecté à $vrai$ (impliqué)
2. $q(x) = \forall$: Si $x_i = vrai$ et $y_i = faux$ alors on doit satisfaire $\mathcal{M}$ sous cette interprétation.

Dans le deuxième cas, en prenant en compte tous les cycles universels de σ, un certain nombre d'interprétations peuvent être considérés comme des modèles de $\mathcal{M}$. Ces interprétations sont collectées dans un ensemble que l'on note $\mathcal{E}(\sigma)$ et que l'on définit ci-dessous.

Définition 73 *Soient $\mathcal{F} = QX\mathcal{M}$ une formule booléenne quantifiée, $\sigma = (x_1, y_1), \dots, (x_n, y_n)$ une symétrie de $\mathcal{F}$ et $\eta_i = (x_1 = y_1) \wedge \dots, (x_{i-1} = y_{i-1}) \wedge x_i \wedge \neg y_i$. Nous définissons $\mathcal{E}(\sigma)$ comme suit : $\mathcal{E}(\sigma) = \bigcup_{i=1..n} \{\rho | \eta_i(\rho) = vrai, q(x_i) = \forall\}$*

Dans le but d'éliminer les symétries de $\mathcal{F}$, on veut satisfaire les deux cas précédents, i.e, propager les littéraux existentiels grâce au $sbp(\sigma)$ et considérer toutes les interprétations de $\mathcal{E}(\sigma)$ comme modèle de $\mathcal{M}$.

Définition 74 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ une formule booléenne quantifiée et $\sigma = (x_1, y_1), \dots, (x_n, y_n)$ une symétrie de $\mathcal{F}$. Soit $sbp^{\exists}(\sigma) = \bigcup_{i=1..n}\{sbp(\sigma) \downarrow X_i | \mathcal{Q}_i = \exists\}$*
on définit : $\mathcal{F}^m = QX[\mathcal{M} \wedge sbp^{\exists}(\sigma)] \vee \mathcal{E}(\sigma)$

Propriété 21 *$\mathcal{F}^m$ et $\mathcal{F}$ sont équivalentes pour la validité.*

Illustrons cette construction en considérant la formule $\mathcal{F}$ de l'exemple 20.

Exemple 28

$$\mathcal{F} = \forall x_1 x_2 \exists x_3 x_4 \left\{ \begin{array}{lllll} (\neg x_1 \vee \neg x_2 \vee x_3) & \wedge & (\neg x_1 \vee \neg x_2 \vee x_4) & \wedge \\ (x_1 \vee \neg x_3 \vee x_4) & \wedge & (x_2 \vee x_3 \vee \neg x_4) & \wedge \end{array} \right.$$

$\sigma = \{(x_1, x_2)(x_3, x_4)\}$ est une symétrie de $\mathcal{F}$. Dans la figure 7.2, comme x_1 et x_2 sont universels, lorsque x_1 est à $vrai$ et x_2 à $faux$ alors le solveur arrête la recherche sur cette branche en déclarant $\mathcal{F}^m$ comme satisfaite grâce à l'ajout du modèle $\rho_2 = \{x_1, \neg x_2\}$.

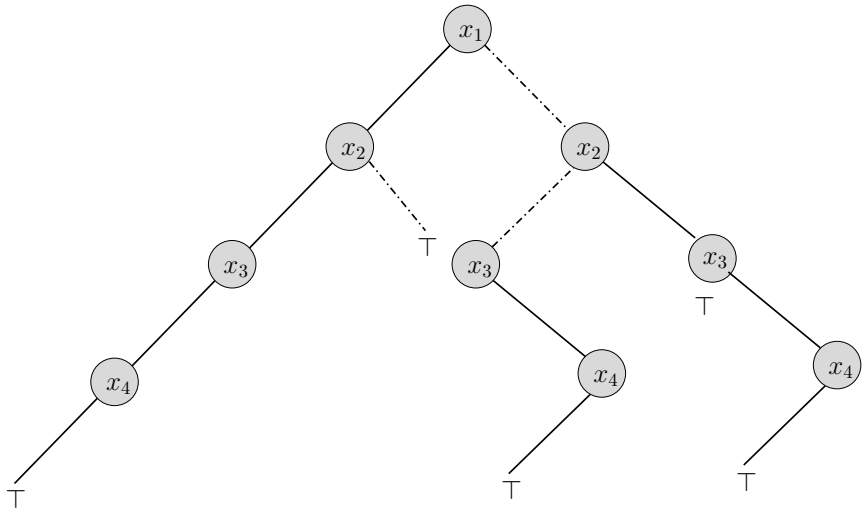


FIG. 7.2 – Arbre de recherche de $\mathcal{F}^m$

On peut noter chaque cube de $\mathcal{E}(\sigma)$ est optimale. En effet, en réordonnant chaque modèle de cet ensemble, on peut remarquer que ces modèles ne contiennent pas de variable existentielle minimale. Pour utiliser cette forme, il faut disposer d'un solveur QBF qui prend en compte une formule de la forme CNF/DNF [ZHANG 2006, SABHARWAL *et al.* 2006, STÉPHAN 2006] ou implémenter une procédure qui intègre les cubes prédéfinis à la lecture de la QBF. Dans la section suivante, nous proposons une autre façon permettant de transformer la forme CNF/DNF obtenue en une formule purement CNF.

7.4.2 M-QSB : Transformation clausale

Dans la propriété 21, nous avons montré que les formules $\mathcal{Q}X\mathcal{M}$ et $\mathcal{F}^m = \mathcal{Q}X[\mathcal{M} \wedge sbp^{\exists}(\sigma)] \vee \mathcal{E}(\sigma)$ sont équivalentes pour la validité. Cependant, transformer $\mathcal{Q}X[\mathcal{M} \wedge sbp^{\exists}(\sigma)] \vee \mathcal{E}(\sigma)$ sous forme CNF peut augmenter considérablement la taille de la formule résultante. Nous proposons de réécrire $\mathcal{F}^m$ sous la forme $\mathcal{Q}X(\mathcal{M} \vee \neg r) \wedge qsbp(\sigma, r)$. Cette nouvelle formule a pour but d'impliquer r à faux lorsqu'on est sur des interprétations symétriques et à vrai pour permettre d'utiliser le $sbp^{\exists}(\sigma)$. Pour réaliser ceci, nous définissons $qsbp(\sigma, r)$ de la manière suivante :

Définition 75 *Soit $\sigma = (x_1, y_1) \dots (x_u, y_u) \dots (x_n, y_n)$ une symétrie de $\mathcal{F}$ telle que $\forall i > u$, (x_i, y_i) est un cycle existentiel (u le rang du dernier cycle universel de σ). Soit :*
$\alpha_i = (x_1 = y_1) \wedge \dots (x_{i-1} = y_{i-1})$.
La fonction $qsbp(\sigma, r) = \bigcup_{1 \leq i \leq n} qsbp_{\sigma_i}$ tel que σ_i est définie comme suit :

1. $1 \leq i < u$
   - $\mathcal{Q}(x_i) = \forall$

$$qsbp(\sigma_i, r) = \left\{ \begin{array}{l} (\neg\alpha_i \vee \neg x_i \vee y_i \vee \neg r) \\ (\neg\alpha_i \vee x_i \vee \neg y_i \vee r) \end{array} \right.$$

   - $\mathcal{Q}(x_i) = \exists$

$$qsbp(\sigma_i, r) = \left\{ \begin{array}{l} (\neg\alpha_i \vee \neg x_i \vee y_i) \\ (\neg\alpha_i \vee x_i \vee \neg y_i \vee r) \end{array} \right.$$

2. $i = u$

$$qsbp(\sigma_i, r) = \left\{ \begin{array}{l} (\neg\alpha_u \vee \neg x_u \vee y_u \vee \neg r) \\ (\neg\alpha_u \vee x_u \vee r) \\ (\neg\alpha_u \vee \neg(x_u \leftrightarrow y_u) \vee r) \end{array} \right.$$

3. $i > u$

$$qsbp(\sigma_i) = \{ \quad (\neg\alpha_i \vee \neg x_i \vee y_i)$$

Finalement nous avons la proposition suivante.

Proposition 9 *Soit $\mathcal{F} = QX\mathcal{M}$ une QBF et σ une symétrie de $\mathcal{F}$, alors $\mathcal{F}$ et $\mathcal{F}^m = QX\exists r\ (\mathcal{M} \vee \neg r) \wedge qsbp(\sigma, r)$ sont équivalentes pour la validité.*

Proposition 10 *Soit $\mathcal{F} = QX\mathcal{M}$ une QBF, σ une symétrie de $\mathcal{F}$ et $\mathcal{F}^m = QX\exists r\ (\mathcal{M} \vee \neg r) \wedge qsbp(\sigma, r)$ La nouvelle formule QBF obtenue en supprimant la symétrie σ. Le nombre des clauses ajoutées (respectivement variables) est en $O(5|\sigma|)$ (respectivement $O(|\sigma| + 1)$) où $|\sigma|$ est le nombre de cycles de σ.*

Grâce à l'introduction de nouvelles variables quantifiées existentiellement, la taille du $sbp(\sigma)$ classique est linéaire en la taille de la symétrie. Dans le pire des cas, pour une symétrie totalement universelle $\sigma = (x_1, y_1) \dots (x_n, y_n)$, on introduit une variable et quatre clauses par cycle. Par conséquent, la taille de $qsbp(\sigma, r)$ est en $O(5n)$.

Exemple 29 *Considérons la QBF*
$\mathcal{F} = \forall x_1, y_1, x_2, y_2 \exists x_3, y_3 \mathcal{M}$ *et* $\sigma = (x_1, y_1)(x_2, y_2)$ *une symétrie de* $\mathcal{F}$.
$\mathcal{F}^m = \forall x_1, y_1, x_2, y_2 \exists \alpha_1, \alpha_2, r\ (\mathcal{M} \vee \neg r) \wedge qsbp(\sigma, r)$. *L'expression de* $qsbp(\sigma, r)$ *est la suivante :*

$$qsbp(\sigma, r) = \left\{ \begin{array}{llll} (x_1 \vee y_1 \vee \alpha_1) & \wedge & (\neg\alpha_1 \vee x_2 \vee y_2 \vee \alpha_2) & \wedge \\ (\neg x_1 \vee \neg y_1 \vee \alpha_1) & \wedge & (\neg\alpha_1 \vee \neg x_2 \vee \neg y_2 \vee \alpha_2) & \wedge \\ (\neg x_1 \vee y_1 \vee \neg r) & \wedge & (\neg\alpha_1 \vee \neg x_1 \vee \neg x_2 \vee y_2 \vee \neg r) & \wedge \\ (x_1 \vee \neg y_1 \vee r) & \wedge & (\neg\alpha_1 \vee y_1 \vee \neg x_2 \vee y_2 \vee \neg r) & \wedge \\ (\neg\alpha_1 \vee \neg\alpha_2 \vee r) & \wedge & (\neg\alpha_1 \vee x_2 \vee \neg r) & \wedge \end{array} \right.$$

la variable α_1 (respectivement α_2) est introduite pour exprimer que $(x_1 = y_1)$ (respectivement. $(x_1 = y_1) \wedge (x_2 = y_2)$).

Exemple 30 *Soient $\mathcal{F} = \exists x_1, y_1 \forall x_2, y_2 \exists x_3, y_3 \mathcal{M}$ une formule QBF et σ une symétrie de $\mathcal{F}$, $\sigma = (x_1, y_1), (x_2, y_2)$. En utilisant la formule $qsbp(\sigma, r)$, les différentes valeurs de r en fonction des variables de l'ensemble $\{x_1, x_2, y_1, y_2\}$ sont représentées dans la figure 7.3.*

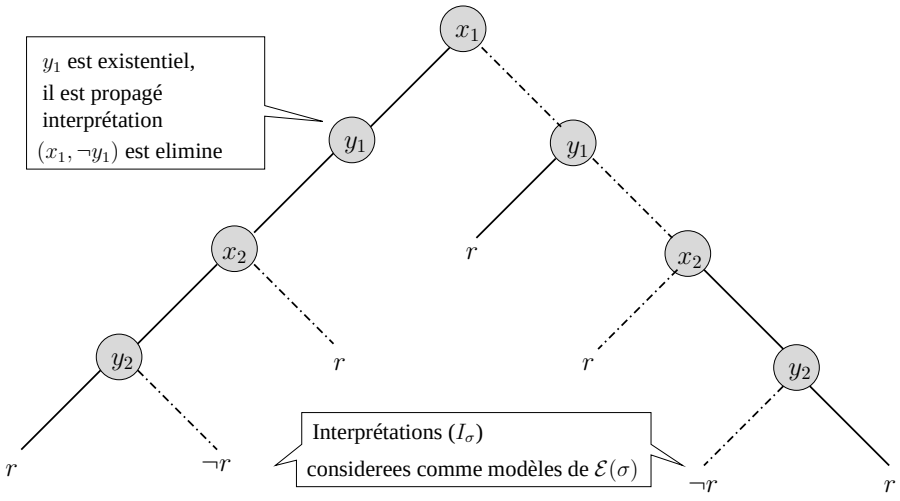


FIG. 7.3 – Arbre de recherche (exemple 30)

Dans ce qui suit, on va montrer comment généraliser la transformation précédente à un ensemble arbitraire de symétries.
Soit $\mathcal{F}$ une QBF et $S = \{\sigma^1 \dots \sigma^n\}$ l'ensemble de ces symétries. On associe pour chaque symétries σ^i une variable unique r^i pour l'éliminer. La suppression de l'ensemble des symétries de S est obtenue en générant une formule QBF équivalente définie ci-dessous.

Définition 76 *Soient $\mathcal{F}$ une QBF et $S = S_\forall \cup S_\exists$ l'ensemble de ces symétries tel que $S_\forall$ (respectivement $S_\exists$) désigne l'ensemble des symétries universelles (respectivement existentielles). Nous définissons*

$$\mathcal{F}^m = \mathcal{Q}X\exists r\ (\mathcal{M} \vee \neg r) \wedge \text{QSBP}(S_\forall, r) \wedge \text{SBP}(S_\exists)$$

- $R = \bigcup_{\sigma^i \in S_\forall} r^i$
- $r = \bigwedge_{\sigma^i \in S_\forall} (r^i)$
- QSBP $(S_\forall, r) = \bigwedge_{\sigma^i \in S_\forall} qsbp(\sigma^i, r^i)$
- SBP $(S_\exists)$ *est le* SBP *classique correspondant à* $S_\exists$.

Proposition 11 *Soient $\mathcal{F}$ une QBF et S l'ensemble de ses symétries. $\mathcal{F}$ est satisfiable si et seulement si $\mathcal{F}^m$ est satisfiable.*

Dans la suite, on propose quelques simplifications utiles pour notre transformation et qui sont utilisées lors de notre évaluation expérimentale.

Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ une QBF et $S = \{\sigma^1 \dots \sigma^p\}$ l'ensemble des symétries de $\mathcal{F}$ tel que $\mathcal{V}(S) \subseteq \{X_i \cup \dots \cup X_n\}$. Si c est une clause de $\mathcal{M}$ tel que $\mathcal{V}(c) \subseteq X_1 \cup \dots \cup X_{i-1}$ i.e, $\mathcal{V}(c) \cap \mathcal{V}(S) = \emptyset$ alors il n'est pas nécessaire d'ajouter $\neg r$ à c. En effet, si I est une affectation de toutes les variables de $\bigcup_{1 \leq j \leq i-1} X_j$ alors $\forall \sigma \in S$, σ est une symétrie de $\mathcal{F}(I)$.

La formule $\mathcal{F}^m$ obtenue permet d'éviter des affectations isomorphes. Néanmoins, l'ajout de $\neg r$ à chaque clause de $\mathcal{M}$ empêche quelques propagations utiles. Par exemple, si $(x \vee y)$ est une clause alors $(x \vee y \vee \neg r)$ est la nouvelle clause de $\mathcal{F}^m$. Lorsque x est affecté à $faux$, alors y est propagé dans $\mathcal{F}$, ce qui n'est pas le cas dans $\mathcal{F}^m$. En outre, dans le pire des cas, si y est affecté à $faux$, $\neg r$ est propagé. Le solveur peut alors passer beaucoup de temps à vérifier l'inconsistance de la sous-formule restante avant de réfuter l'affectation de y à $faux$.
L'ajout de $\neg r$ à toutes les clauses de $\mathcal{M}$ peut donc rendre $\mathcal{F}^m$ plus difficile à résoudre que l'instance originale. Le temps d'exécution dans plusieurs cas passe de quelques millièmes de secondes à quelques milliers. Pour Quantor [BIERE 2004] par exemple, qui est basé sur l'extension et la résolution, la mémoire vive se retrouve vite saturée du fait que r appartient au groupe de quantificateurs le plus interne. Le solveur sKizzo [BENEDETTI 2004] arrive légèrement mieux à s'en sortir. Quant aux solveurs basés sur la recherche, l'ajout de $\neg r$ à toutes les clauses bloque, on l'a vu, des propagations utiles et rend donc la résolution plus complexe que prévu.

Mais revenons à notre transformation. Le but est de propager $\neg r$ à $vrai$ lorsque l'on est sur une branche symétrique et d'arrêter ainsi la recherche en considérant que nous avons trouvé un modèle. Nous pouvons donc restreindre le nombre de clauses que l'on considère valides sans changer la validité de la formule. Ce qui veut dire que l'on peut ajouter $\neg r$ à un sous-ensemble des clauses originales. La question qui suit, est de savoir à quelles clauses on va ajouter $\neg r$. La réponse à cette question n'est pas évidente.
Nous avons choisi de n'ajouter $\neg r$ qu'aux clauses non binaires pour permettre, a priori, de faciliter la propagation.

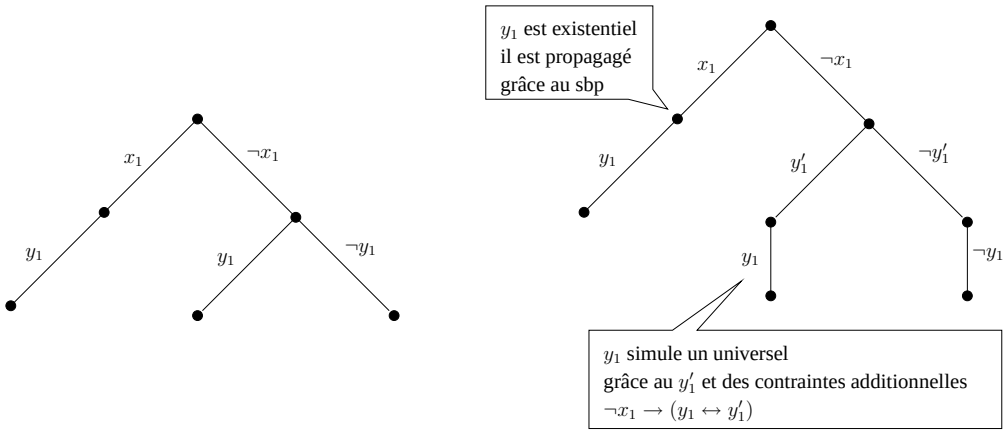
D'autres limitations concernent les littéraux monotones (qui n'apparaissent que positivement ou négativement dans la formule initiale). Considérons une formule QBF $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ avec une symétrie $\sigma = (x_1, y_1) \dots (x_n, y_n)$. Supposons qu'il existe un cycle (x_m, y_m) de σ tel que x_m et y_m sont monotones. Dans ce cas, en supprimant σ, les deux littéraux x_m et y_m ne sont plus monotones dans la nouvelle formule générée. Comme les littéraux monotones jouent un rôle important dans la résolution des QBFs [GIUNCHIGLIA *et al.* 2004], ceci peut avoir un impact sur l'efficacité des solveurs. Pour contourner ce problème, nous proposons les deux possibilités suivantes. La première, consiste à éliminer le cycle (x_m, y_m) de σ et à propager x_m et y_m suivant leurs quantification. La seconde consiste à réduire la symétrie σ aux sous-ensembles $\{(x_1, y_1) \dots (x_{m-1}, y_{m-1})\}$. Nous avons opté pour cette seconde option dans nos expérimentations. Finalement, si σ est de la forme $(x, \neg x)$ telle que x est universel, alors x est requantifié existentiellement et la clause $c = (\neg x)$ est ajoutée conjonctivement à la matrice.

7.5 Approche basée sur les nogoods (N-QSB)

Dans cette section, nous introduisons la seconde approche, basée elle sur les nogoods, utilisée pour supprimer les symétries d'une formule QBF. Contrairement à l'approche basée sur les modèles, les affectations symétriques associées aux littéraux universels sont considérées comme des nogoods de la formule QBF. Pour cela, en plus du SBP classique utilisée pour supprimer les symétries existentielles (voir la proposition 5, de nouvelles contraintes SBP (nommés QSBP) sont générées pour les symétries universelles. En d'autres termes, la QBF originale est réécrite de telle manière que les littéraux universels peuvent être impliqués pour éliminer des affectations symétriques. La nouvelle formule QBF est construite en chan-

geant les quantifications de certaines variables universelles et en ajoutant de nouvelles variables. Ces variables modifiées vont être associées à de nouvelles variables universelles et leur relation sera décrite dans le QSBP généré. De plus, pour garder l'équivalence entre la formule initiale et la nouvelle formule asymétrique, un nouvel ordre du préfixe sera calculé.

Commençons, tout d'abord, par illustrer cette approche avec une formule QBF contenant une seule symétrie universelle (x_1, y_1). Notre but est d'éliminer l'interprétation $\{\dots x_1, \neg y_1 \dots\}$ qui est symétrique à $\{\dots \neg x_1, y_1 \dots\}$ comme le montre la figure 7.4.a. En d'autres termes, quand $x_1 = vrai$ (cas 1) nous forçons y_1 à être vrai. Comme y_1 est universellement quantifié cela impliquera une inconsistance. Pour rendre cette implication possible, nous proposons de changer la quantification de y_1 en la rendant existentielle. De plus, quand x_1 est faux (cas 2), les deux affectations $y_1 = vrai$ et $y_1 = faux$ doivent être considérées. Pour rendre possible ces deux cas, nous introduisons une nouvelle variable y'_1. Cette variable joue le même rôle que y_1 dans le cas 2, alors qu'elle ne sert à rien dans le cas 1. Ceci est réalisé grâce à la contrainte $\neg x_1 \rightarrow (y_1 \leftrightarrow y'_1)$ qui est donc le QSBP de la symétrie. La figure 7.4.b illustre le comportement de l'approche N-QSB.



a. Comportement espéré b. Comportement réel

FIG. 7.4 – L'approche $N - QSB$: un exemple illustratif

Nous commençons par décrire formellement cette approche pour le cas d'une seule symétrie, puis nous généralisons aux cas des symétries quelconques.

7.5.1 N-QSB : Élimination d'une seule symétrie

Définition 77 *Soient $\mathcal{F}$ une formule QBF, σ une symétrie de $\mathcal{F}$ et $c = (x, y)$ un cycle σ. Nous nommons x (resp. y) le in-littéral de c (resp. out-littéral).*

Comme nous l'avons montré dans l'introduction de l'approche, dans le cas d'une symétrie universelle, les out-littéraux peuvent voir leur valeur imposée par le SBP. Pour contourner ce problème, les out-littéraux vont être requantifiés existentiellement avec des rangs supérieurs à ceux des in-littéraux associés dans les cycles. La définition suivante formalise celà.

Définition 78 *Soit $\sigma = \{(x_1, y_1) \dots (x_k, y_k) \dots (x_n, y_n)\}$ avec $1 \leq k \leq n$ une symétrie universelle p-ordonnée. Nous définissons* QSBP $(\sigma) = \cup\{qsbp(\sigma(y_k)), 1 \leq k \leq n, y_k$ *universel*$\}$ *comme le* QSBP *associé à σ. Le* QSBP (σ) *est construit en utilisant les deux étapes suivantes :*

1. *Ajout de variables auxiliaires : pour chaque cycle universel $(x_i, y_i) \in \sigma$, on associe une nouvelle variable universelle y'_i au out-littéral y_i. Le quantificateur universel de y_i est substitué par un quantificateur existentiel.*

2. *Génération de nouvelles contraintes :*
   - *Si (x_1, y_1) est un cycle universel alors $qsb(\sigma(y_1)) = \{\neg x_1 \rightarrow (y_1 \leftrightarrow y'_1)\}$*
   - *Pour tout $k > 1$, si (x_k, y_k) est un cycle universel alors $qsbp(\sigma(y_k))$ contient les contraintes suivantes :*
     - *$\neg x_k \rightarrow (y_k \leftrightarrow y'_k)$ si $x_k \neq \neg y_k$*
     - *$((\neg x_j \wedge y_j) \rightarrow (y_k \leftrightarrow y'_k)), \forall j$ tel que $1 \leq j < k$*

Exemple 31 *Soit $\sigma = (x_1, x_2)$ une symétrie d'une formule QBF $\mathcal{F}$ telle que $q(x_1) = q(x_2) = \forall$. D'après la définition du* QSBP*, le quantificateur de x_2 est substitué par un quantificateur existentiel. On associe à x_2 une nouvelle variable x'_2. $qsbp(\sigma(x_2))$= $(\neg x_1 \rightarrow (x_2 \leftrightarrow x'_2))$. On peut remarquer aussi que $sbp(\sigma(x_2)) \wedge qsbp(\sigma(x_2)) \leftrightarrow (x_2 = (x_1 \vee x'_2))$*

Pour une symétrie $\sigma = \{(x_1, y_1) \dots (x_i, y_i) \dots (x_n, y_n)\}$, nous associons pour chaque variable y_i, telle que $q(y_i) = \forall$, une variable y'_i. Entre les variables x_i, y_i et y'_i on définit un nouvel ordre. La définition suivante en fait état.

Définition 79 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_i X_i \dots \mathcal{Q}_n X_n \, \mathcal{M}$ une QBF, $\sigma = \sigma_1 \dots \sigma_i \dots \sigma_m$ telles que $\forall i \in \{1 \dots m\}$ une symétrie universelle p-ordonnée. Soit j tel que $Q_j = \forall$ et $\sigma_j = \sigma[X_j] = (x_1, y_1) \dots (x_n, y_n)$ et $Y' = \{y'_1 \dots y'_n\}$ l'ensemble des variables ajoutées. Nous définissons un nouveau rang de $Var(\sigma_j) \cup Y'$ comme suit : $\forall (x_k, y_k) \in \sigma_j$,*

- *$rang(x_k) < rang(y'_k) < rang(y_k)$ tel que $1 \leq k \leq n$ et*

Cette relation qui vient d'être introduite peut être exprimée sous forme d'un graphe orienté appelé graphe de précédence.

Définition 80 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_i X_i \dots \mathcal{Q}_n X_n \, \mathcal{M}$ une QBF, $\sigma = \{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ une symétrie universelle et p-ordonnée tel que $i \in \{1 \dots m\}$ et soit $\sigma_j = \sigma[X_j] = \{(x_1, y_1), \dots (x_n, y_n)\}$ tel que $Q_j = \forall$. Nous définissons $\mathcal{G}_{\sigma_j}(\mathcal{V}, \mathcal{A})$ le graphe de précédence associé à σ_j avec $Q_j = \forall$ comme suit :*

- *$\mathcal{V} = \bigcup_{1 \leq k \leq n} \{x_k, y_k, y'_k\}$*
- *$\mathcal{A} = \bigcup_{1 \leq k \leq n} \{(x_k, y'_k), (y'_k, y_k)\}$*

L'étape suivante consiste à transformer le quantificateur $Q_j X_j$ afin de respecter la définition du SBP. Ce dernier doit respecter les conditions sur l'ordre que l'on vient d'introduire liant les anciennes variables du groupe X_j aux variables auxiliaires. La méthode du tri topologique est utilisée à cette fin.

Dans le cas d'une symétrie singulière, un tel graphe est acyclique. La figure 7.5, représente le graphe de précédence de $\sigma_j = \sigma[X_j]$ tel que $Q_j = \forall$ et $\sigma_j = \{(x_1, y_1) \dots (x_n, y_n)\}$. Un ordre possible des variables est le suivant : $[x_1 \dots x_n y'_1 y_1 \dots y'_n y_n]$

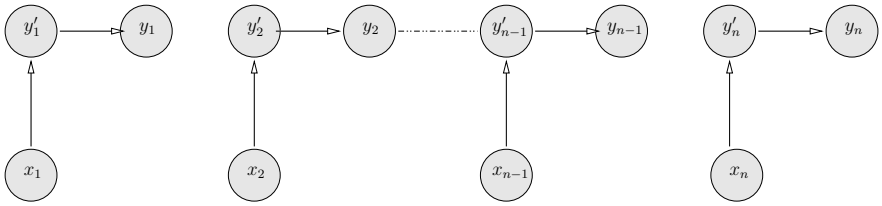


FIG. 7.5 – Graphe de précédence de σ_j

Définition 81 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_i X_i \dots \mathcal{Q}_n X_n \, \mathcal{M}$ une QBF, $\sigma = \{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ une symétrie universelle p-ordonnée. Pour $j \in \{1 \dots m\}$ $\sigma_j = \sigma[X_j] = \{(x_1, y_1), \dots (x_n, y_n)\}$. Chaque groupe de quantificateurs $Q_j X_j$ de $\mathcal{F}$ est réécrit comme suit :*

1. *Si $\mathcal{Q}_j = \forall$ et $\sigma[X_j] \neq \emptyset$*
   - *alors $\mathcal{Q}'_j X'_j = \forall X_j \backslash \mathcal{V}(\sigma_j) x_1 \dots x_n \exists e \forall y'_1 \exists y_1 \dots \forall y'_n \exists y_n$ (voir figure 7.5)*
   - *tel que $X'_j = X_j \cup \{y'_1 \dots y'_n\}$*
2. *sinon $\mathcal{Q}'_j X'_j = \mathcal{Q}_j X_j$ avec $X'_j = X_j$*

$\mathcal{F}$ est donc réécrit comme

$$\mathcal{F}^n = \mathcal{Q}'_1 X'_1 \dots \mathcal{Q}'_i X'_i \dots \mathcal{Q}'_n X'_n \, \mathcal{M} \wedge \text{SBP}(\sigma) \wedge \text{QSBP}(\sigma)$$

La variable e est ajoutée pour contraindre l'ensemble $\{x_1 \dots x_n\}$ à être affecté avant l'ensemble $\{y'_1 \dots y'_n\}$ dans le premier cas de la définition 81.

Illustrons cette construction en considérant la formule $\mathcal{F}$ de l'exemple 20. La nouvelle QBF est définie par :
$\mathcal{F}^n = \forall x_1 \forall x'_2 \exists x_2 x_3 x_4 \, \mathcal{M}^n$ avec :

$$\mathcal{M}^n = \left\{ \begin{array}{lllll} (\neg x_1 \vee \neg x_2 \vee x_3) & \wedge & (\neg x_1 \vee x_2) & \wedge \\ (\neg x_1 \vee \neg x_2 \vee x_4) & \wedge & (\neg x_1 \vee \neg x_3 \vee x_4) & \wedge \\ (x_1 \vee x_3 \vee \neg x_4) & \wedge & (x_2 \vee \neg x_1 \vee x_4) & \wedge \\ (x_2 \vee \neg x_3 \vee x_4) & \wedge & (x_1 \vee \neg x_2 \vee x'_2) & \wedge \\ & & (x_1 \vee x_2 \vee \neg x'_2) & \end{array} \right.$$

Nous imposons entre les variables x_1, x_2 et x'_2 la relation de précédence suivante :
$rang(x_1) < rang(x'_2) < rang(x_2)$. La figure 7.6 représente l'arbre de recherche de la nouvelle QBF $\mathcal{F}^n$.
Lorsque x_1 est affecté à vrai, à partir de la clause $(\neg x_1 \vee x_2)$ on déduit que x_2 est vrai aussi. Si x_1 est affecté à faux, le quantificateur original de x_2 est déduit par substitution. Ceci est garanti grâce à l'ajout de la contrainte $\neg x_1 \rightarrow (x_2 \leftrightarrow x'_2)$ qui peut s'écrire sous la forme de conjonction des deux clauses :$(x_1 \vee \neg x_2 \vee x'_2) \wedge (x_1 \vee x_2 \vee \neg x'_2)$ exprimant le fait que lorsque x_1 est affecté à faux, x_2 et x'_2 deviennent équivalents. Comme le rang de x'_2 quantifié universellement est inférieur au rang de x_2 quantifié quant à lui existentiellement, on peut tout simplement remplacer les occurrences de x_2 par x'_2 et ôter x_2 du préfixe. Ce qui permet de retrouver le x_2 (représenté par x'_2) d'origine. La combinaison du SBP et du QSBP permet de préserver la validité de la formule d'origine.

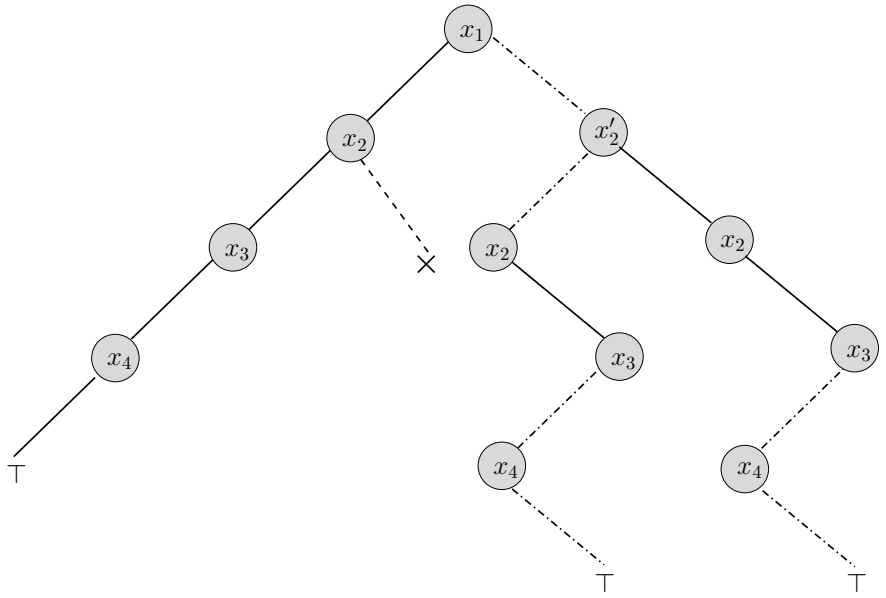


FIG. 7.6 – Arbre de recherche $\mathcal{F}^n$ de l'exemple 20

Propriété 22 *Soit $\mathcal{F}$ une QBF et σ une symétrie, alors $\mathcal{F}$ est valide si et seulement si $\mathcal{F}^n$ est valide.*

Preuve : La QBF $\mathcal{F}^n$ est obtenue en ajoutant le SBP correspondant aux symétries existentielles, et le SBP et le QSBP pour les symétries universelles conjonctivement à la formule QBF originale et en réécrivant son préfixe. Pour montrer que les deux QBFs sont équivalentes il suffit de montrer que la nouvelle formule respecte exactement la définition du SBP. Pour les symétries existentielles le résultat est immédiat. Considérons donc que notre symétrie est universelle et p-ordonnée.

Soit $\sigma = \{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ telle que $\{\sigma_1 \dots \sigma_{j-1}\}$ existentielles et σ_j universelle avec $\{\sigma_1 \dots \sigma_{j-1}\} = \{(x_1, y_1) \dots (x_{k-1}, y_{k-1})\}$ et $\sigma_j = \{(x_k, y_k) \dots (x_n, y_n)\}$. Nous distinguons deux cas :

- $\forall\ (1 \leq i \leq (k-1)$ on a : $x_i = y_i$.
  Dans ce cas notre symétrie nous permet encore d'effectuer des coupures dans l'arbre de recherche. Après avoir affecté l'ensemble $X'_k \backslash Var(\sigma_k)$, on affecte l'ensemble $\{x_k \dots x_n\}$. Supposons que $x_k = \dots x_{k+s-1} = vrai$ et $x_{k+s} = faux$. D'après la définition du SBP, $y_k = \dots y_{k+s-1} = vrai$. Puisque $x_{k+s} = faux$, d'après la définition du QSBP, $y'_{k+s} \leftrightarrow y_{k+s}$ et on peut remplacer les occurrences de y_{k+s} par y'_{k+s}.
  - si $y'_{k+s} = vrai$ le SBP devient vrai (ne permet plus de couper des branches) et en utilisant la définition du QSBP $\forall j > (k+s) \mid y'_j \leftrightarrow y_j$. Dans ce cas $\mathcal{Q}'_j X'_j = \mathcal{Q}_j X_j$.
  - Si $y'_{k+s} = faux$ on passe aux valeurs affectés à $x_{k+s+1} \dots x_n$
- $\exists (x_{j_1}, y_{j_1}) \mid (k-1) \leq j_1 \leq n$ et $(x_{j_1} \neq y_{j_1})$.
  - si $x_{j_1} = faux$ et $y_{j_1} = vrai$. Le SBP devient Vrai et d'après le QSBP, $y'_{j_1} \leftrightarrow y_{j_1}$ pour tout $i > (j-1)$. Dans ce cas $\mathcal{Q}'_j X'_j = \mathcal{Q}_j X_j$.
  - si $x_{j_1} = vrai$ et $y_{j_1} = faux$, dans ce cas il existe $j_0 \leq j_1$ tel que $x_{j_0} = faux$ et $y_{j_0} = vrai$. Dans ce cas d'après le QSBP pour tout $k \leq i \leq n$, $y'_i \leftrightarrow y_i$ et $\mathcal{Q}'_j X'_j = \mathcal{Q}_j X_j$. □

7.5.2 N-QSB : Élimination d'un ensemble de symétries

Dans la section 7.5.1, la transformation d'une formule ne contenant qu'une seule symétrie est explicitée. Quand une formule QBF contient plusieurs symétries, le traitement se complique du fait que ces

symétries ne peuvent être supprimées les unes indépendamment des autres en utilisant l'approche décrite dans la section précédente. Les interactions entre ces différentes symétries nous imposent d'effectuer des modifications qui les prennent en compte.

Illustrons ces interactions entre les symétries en considérant l'exemple d'une formule QBF $\mathcal{F}$ ayant deux symétries σ et σ' telles que $\sigma[X_1] = \{(x_1, x_4)\}$ et $\sigma'[X_1] = \{(x_3, x_4)\}$ avec $\mathcal{Q}_1 = \forall$. Le QSBP généré à partir de la première (resp. deuxième) symétrie est $\neg x_1 \rightarrow (x_4 \leftrightarrow x'_4))$ (resp. $\neg x_3 \rightarrow (x_4 \leftrightarrow x'_4)$)). Nous pouvons voir, que les deux symétries partagent le même out-littéral x_4. Comme expliqué précédemment, pour un cycle universel donné, le QSBP exprime les conditions sous lesquelles le out-littéral et son littéral auxiliaire correspondant sont équivalents. En considérant les deux symétries σ et σ', on peut constater aisément que x'_4 est équivalent à x_4 seulement si aucune des deux symétries ne peut impliquer x_4. Pour l'ensemble des deux symétries, le QSBP est donc donné par la formule : $(\neg x_1 \wedge \neg x_3) \rightarrow (x_4 \leftrightarrow x'_4))$ qui peut être vue comme une `corrélation` entre les deux symétries.

Dans le cas général, en présence de plusieurs symétries, un littéral est équivalent à son correspondant seulement si ce littéral ne peut être impliqué par aucune des symétries.

Définition 82 *Soit $\mathcal{F}$ une QBF, et*
$\sigma = \{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ *et* $\sigma' = \{\sigma'_1 \dots \sigma'_i \dots \sigma'_m\}$
deux symétries de $\mathcal{F}$. Soit $k \in \{1 \dots m\}$ tel que

- $\mathcal{Q}_i = \forall$,
- $(x_k, y_k) \in \sigma[X_i]$,
- $(x'_k, y'_k) \in \sigma'[X_i]$,
- $y_k = y'_k = y_{k_0}$,
- $qsbp(\sigma(y_{k_0})) = \{\alpha_1 \rightarrow (y_{k_0} \leftrightarrow y'_{k_0})) \dots \alpha_N \rightarrow (y_{k_0} \leftrightarrow y'_{k_0})\}$ *et*
- $qsbp(\sigma'(y_{k_0})) = \{\beta_1 \rightarrow (y_{k_0} \leftrightarrow y'_{k_0})) \dots \beta_M \rightarrow (y_{k_0} \leftrightarrow y'_{k_0})\}$.

Nous définissons l'opérateur binaire η exprimant la `corrélation` entre σ et σ' par rapport à y_{k_0} comme suit :

$$\eta(\sigma(y_{k_0}), \sigma'(y_{k_0})) = \begin{cases} \{(\alpha_1 \wedge \beta_1) \rightarrow (x \leftrightarrow x') & \dots \quad (\alpha_1 \wedge \beta_M) \rightarrow (y_{k_0} \leftrightarrow y'_{k_0}) \\ \dots \\ \dots \\ (\alpha_N \wedge \beta_1) \rightarrow (y_{k_0} \leftrightarrow y'_{k_0}) & \dots \quad (\alpha_N \wedge \beta_M) \rightarrow (y_{k_0} \leftrightarrow y'_{k_0}) \end{cases}$$

Définition 83 *Soit $\mathcal{S} = \{\sigma^1 \dots \sigma^n\}$ l'ensemble des symétries d'une QBF. Pour une variable $y \in X_k$ telle que $\mathcal{Q}_k = \forall$ Nous définissons :*

- $\mathcal{S}[y] = \{\sigma^j, \exists x \in X_k$ *tel que* $(x, y) \in \sigma^j[X_k]\} = \{\sigma'^1 \dots \sigma'^{|\mathcal{S}[y]|}\}$.
- QSBP $(\mathcal{S}[y]) = \eta(\eta \dots \eta(qsbp(\sigma'^1), \sigma'^2), \dots, \sigma'^{|\mathcal{S}[y]|}) \dots)$.
- $\mathcal{V}(S) = \bigcup(\mathcal{V}(\sigma^i), 1 \leq i \leq n)$

– $\mathcal{V}(\mathcal{S}^{\forall}) = \bigcup(x \in \mathcal{V}(S), q(x) = \forall)$

Propriété 23 *Soit $\mathcal{S} = \{\sigma^1 \dots \sigma^n\}$ l'ensemble des symétries d'une QBF $\mathcal{F} = Q\ \mathcal{M}$. La nouvelle matrice de $\mathcal{M}^n$ est définie comme suit :*

$$\mathcal{M} \wedge (\bigwedge_{1 \leq i \leq n} \textsc{Sbp}(\sigma^i)) \wedge (\bigwedge_{y \in \mathcal{V}(\mathcal{S}^{\forall})} \textsc{QSbp}(\mathcal{S}[y]))$$

7.5.3 N-QSB : Transformation du préfixe

Dans la section 7.5.1, nous avons vu comment réécrire le préfixe d'une formule QBF contenant une seule symétrie. Dans le cas de plusieurs symétries, le processus consiste à construire le graphe de précédence correspondant à l'ensemble des symétries et à appliquer le tri topologique pour déterminer un ordre possible des variables. Ceci est illustré dans l'exemple 32.

Exemple 32 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n\ \mathcal{M}$ une QBF, σ et σ' deux symétries de $\mathcal{F}$ telles que :*

– $\mathcal{Q}_1 = \forall$,

– $\sigma[X_1] = \{(x_1, x_2)(x_3, x_4)(x_5, x_6)\}$ *et*

– $\sigma'[X_1] = \{(x_1, x_3)(x_2, x_5)(x_4, x_6)\}$

Le graphe de précédence associé à σ et σ' est représenté dans la figure 7.7.

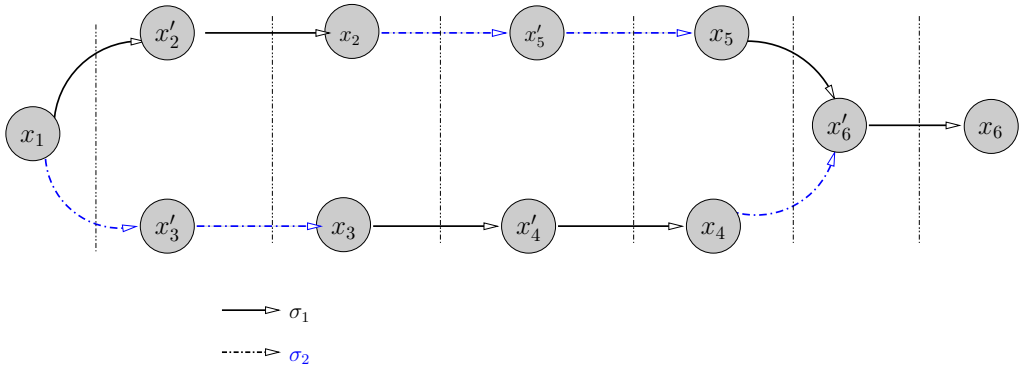


FIG. 7.7 – Graphe de précédence de $\sigma_1[X_1]$ et $\sigma_2[X_1]$ (exemple 32)

Le quantificateur $\mathcal{Q}'_1 X'_1$ est réécrit sous forme de : $\forall x_1 \exists \alpha \forall x'_2 x'_3 \exists x_2 x_3 \forall x'_4 x'_5 \exists x_4 x_5 \forall x'_6 \exists x_6$

7.5.4 Out-littéraux positifs et négatifs

Pour préserver l'équivalence (pour la validité) entre la formule QBF originale et celle construite, regardons maintenant le dernier problème dû aux interactions entre les différentes symétries. Comme illustré dans l'exemple suivant, le problème se pose quand une variable apparaît comme un out-littéral positif (x) dans une symétrie et négatif ($\neg x$) dans une autre.

Exemple 33 *Soient $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_i X_i \dots \mathcal{Q}_n X_n \mathcal{M}$ une QBF, σ et σ' deux symétries de $\mathcal{F}$ telles que $\sigma[X_i] = \{(x_1, x_4)\}$ et $\sigma'[X_i] = \{(x_2, \neg x_4)\}, \mathcal{Q}_i = \forall$.*

En utilisant notre approche, le groupe de quantificateurs $\mathcal{Q}_i X_i$ est réécrit sous la forme $\forall(X_i \backslash \mathcal{V}(\sigma, \sigma'))\forall x_1 x_2\ \exists\alpha\forall x_4'\exists x_4$. Le $sbp(\sigma)$ généré contient la clause $c = (\neg x_1 \vee x_4)$. Pour σ' son $sbp(\sigma')$ correspondant contient la clause $c' = (\neg x_2 \vee \neg x_4)$. La variable universelle x_4 est substituée par une autre quantifiée quant à elle existentiellement. En appliquant la règle de la Q-résolution [BÜNING *et al.* 1995] entre c et c' on obtient une clause $r = (\neg x_1 \vee \neg x_2)$ totalement universelle. Par conséquent, la formule QBF obtenue est insatisfiable. Trouver un nouvelle ordre qui permet d'éviter ce problème est une tâche très difficile. Dans l'exemple 33, un tel problème peut être évité en considérant par exemple l'ordre $x_1 < x_4 < x_2$ au lieu de l'ordre lexicographique. Une autre possibilité consiste à appliquer l'opération de composition entre les symétries. Plus précisément, en considérant la nouvelle symétrie $\sigma'' = \sigma \circ \sigma' \circ \sigma = \{(x_1, \neg x_2)\}$ obtenue par l'opération de composition. Si en plus de σ et σ', on considère aussi σ'', alors la résolvante précédente générée en utilisant la Q-résolution n'est plus totalement universelle. En effet, comme x_2 est un out-littéral, son quantificateur universel est substitué par un nouveau quantificateur existentiel. Donc la résolvante r contient le littéral $\neg x_2$ re-quantifiée existentiellement. Finalement, le groupe de quantificateurs $\mathcal{Q}_i X_i$ peut être réécrit sous la forme $\mathcal{Q}_i' X_i' = \forall x_1 \exists\alpha\forall x_2'\exists x_2\forall x_4'\exists x_4$ à partir du graphe de précédence associé à σ, σ' et σ''.

La discussion précédence nous donne une idée de la manière de résoudre dans le cas général le problème posé par l'ensemble de symétries contenant deux cycles universels de la forme (y, x) et $(z, \neg x)$. Dans les expérimentations, le problème est évité en utilisant la restriction suivante.

Définition 84 *Soit $\mathcal{F} = \mathcal{Q}_1 X_1 \dots \mathcal{Q}_n X_n \mathcal{M}$ une QBF, $y \in \mathcal{V}_\forall(\mathcal{F})$ et $\mathcal{S}$ l'ensemble des symétries de $\mathcal{F}$. Nous définissons $\mathcal{S}[y] \downarrow y = \{\sigma \downarrow y | \sigma \in \mathcal{S}[y]\}$. Pour $\sigma = \{(x_1, y_1), \dots, (x_k, y) \dots, (x_n, y_n)\}$, on définit $\sigma \downarrow y = \{(x_1, y_1), \dots, (x_{k-1}, y_{k-1})\}$. La restriction de $\mathcal{S}$ par rapport à y est définie comme $rt(S, y) = \{\sigma | \sigma \in S, \sigma[y] = \emptyset, \sigma[\neg y] = \emptyset\} \cup \mathcal{S}[y] \cup (\mathcal{S}[\neg y] \downarrow \neg y)$. Pour l'ensemble de variables $\mathcal{V}_\forall(\mathcal{F}) = \{v_1, \dots, v_{|\mathcal{V}_\forall(\mathcal{F})|}\}$, on définit $rt(S, \mathcal{V}_\forall(\mathcal{F})) = rt(\dots rt(S, v_1), v_2) \dots v_{|\mathcal{V}_\forall(\mathcal{F})|}) \dots)$*

Notons que si $S[y] = \emptyset$ ou $S[\neg y] = \emptyset$, alors $rt(S, y) = S$. Naturellement, pour un ensemble de symétries S donnée, la nouvelle formule QBF est générée en utilisant $rt(S, \mathcal{V}_\forall(\mathcal{F}))$. De cette façon, la formule obtenue est équivalente à la formule originale pour la validité.

7.5.5 Complexité

Soit σ une symétrie d'une QBF et $CNF(qsbp(\sigma))$ la CNF représentant $qsbp(\sigma)$. La complexité du pire des cas de $CNF(qsbp(\sigma))$ est en $O(|\sigma|^2)$. Considérons $\sigma = \{\sigma_1, \dots, \sigma_n\}$ avec $\sigma_i = (x_i, y_i)$, $1 \leq i \leq n$, le pire des cas survient quand tous les cycles de σ sont universels. Dans ce cas, $qsbp(\sigma) = \bigcup_{1 \leq i \leq n} qsbp(\sigma_i(y_i))$. $|qsbp(\sigma)| = \sum_{1 \leq i \leq n} |qsbp(\sigma(y_i))|$. $|qsbp(\sigma(y_i))|$ est égale à $2(i-1)+2 = 2i$ (voir définition 78). Donc, $|qsbp(\sigma)| = \sum_{1 \leq i \leq n} 2i$ qui est égale à $n(n+1)$. Plus intéressant encore, en utilisant les mêmes variables introduites pour $CNF(sbp(\sigma))$, la taille du $qsbp(\sigma)$ devient *linéaire*. Malheureusement, à cause des corrélations entre les différentes symétries, le $qsbp(\mathcal{S})$ associé à l'ensemble des symétries de $\mathcal{S}$ est en $O(|\mathcal{S}_\forall|)$ où $|\mathcal{S}_\forall|$ est le nombre des symétries universelles. Comme le nombre de symétries peut être exponentiel, la taille du QSBP est exponentielle dans le pire des cas.

Notons qu'en pratique, dans le SBP classique on introduit des variables auxiliaires pour obtenir une taille linéaire. En utilisant ces mêmes variables additionnelles pour la génération du QSBP, la taille de ce dernier devient linéaire.

7.6 Expérimentations

Les résultats expérimentaux indiqués dans cette section sont obtenus sur un Xeon 3.2 GHz (2 GB RAM) sur un large panel d'instances symétriques disponible sur [GIUNCHIGLIA *et al.* 2001a]. Cet en-

semble d'instances QBF contient différentes familles comme `biu`, `toilet`, `k_*`, `FPGA`, `tip`. Nous réalisons ces expérimentations avec différents solveurs, SQBF [SAMULOWITZ & BACCHUS 2005], SKIZZO [BENEDETTI 2005a],
NCQUBE [GIUNCHIGLIA *et al.* 2006] et SEMPROP[LETZ 2002a]. Nous limitons le temps de calcul à 1800 secondes (30 minutes) et la mémoire vive est limitée quant à elle à 1 giga-octet.

Avant tout, notons que la détection de symétries et le mécanisme d'élimination sont calculés efficacement. La médiane du temps de calcul du SBP (incluant la détection de symétries) est inférieure à 60 secondes. Pour 80% des problèmes il est inférieur à 3 secondes. Seulement 2% des instances passent la limite des 10 secondes.

Commençons par quelques remarques sur les différentes instances. Quelques unes sont assez faciles (environ 150 sont résolues en moins d'une seconde par tous les solveurs (avec ou sans le SBP)) et proche de 100 instances testées sont vraiment très difficiles (non résolues par aucun des solveurs). De plus, la moitié des benchmarks contiennent seulement des symétries existentielles.

Dans la section suivante, on donne un résumé des résultats obtenus par les différents solveurs lors de ces expérimentations.

7.6.1 Résumé des résultats

La table 7.1 indique le nombre d'instances (nb) pour chaque famille. Pour chaque solveur, le nombre d'instances originales ($\mathcal{F}$), le nombre d'instances résolues en utilisant l'approche $N-QSB$ ($\mathcal{F}^n$) et le nombre d'instances résolues par l'approche $M-QSB$ (($\mathcal{F}^m$)) en moins de 1800 secondes sont aussi indiqués.

Excepté pour les 3 familles (`tip*`, `blif*` et `TOILET*`), l'ajout des prédicats d'élimination de symétries n'améliore pas les performances de NCQUBE. Pour tous les autres solveurs, le QSBP améliore les performances des solveurs sur de nombreuses instances. Il semble également d'après les expérimentations que la méthode N-QSB produit de bons résultats par rapport à l'approche M-QSB. Ce résultat n'est pas surprenant dès lors que la transformation sous forme QCNF de l'approche M-QSB ne reflète pas la meilleure utilisation qui peut être faite de cette méthode.

		NCQUBE			SKIZZO			SQBF			SEMPROP		
Familles	nb	$\mathcal{F}$	$\mathcal{F}^n$	$\mathcal{F}^m$	$\mathcal{F}$	$\mathcal{F}^n$	$\mathcal{F}^m$	$\mathcal{F}$	$\mathcal{F}^n$	$\mathcal{F}^m$	$\mathcal{F}$	$\mathcal{F}^n$	$\mathcal{F}^m$
biu	23	**23**	9	11	1	0	**4**	0	0	0	1	**8**	**8**
sort	82	**33**	13	17	29	30	**31**	**15**	12	14	**33**	29	30
lut	8	6	6	6	8	8	8	5	**7**	6	6	**7**	**7**
asp	104	**104**	96	103	76	**90**	82	0	0	0	**104**	**104**	103
tip	125	41	43	**44**	15	**16**	**16**	10	10	10	10	10	10
x	100	99	99	99	70	71	**72**	76	**80**	76	88	88	88
blif_	32	20	**21**	**21**	**21**	**21**	20	16	**18**	**18**	17	**18**	**18**
chain	12	**12**	**12**	7	12	12	12	9	**12**	5	12	12	12
TOILET	8	6	**7**	**7**	8	8	8	**8**	6	**8**	6	6	6
toilet_	111	**111**	109	**111**	111	111	111	**109**	102	**109**	110	**111**	**111**
k_	220	**135**	**135**	134	**187**	**187**	186	106	**108**	105	134	**142**	131
BLOCKS	4	**3**	2	2	3	**4**	3	4	4	4	0	0	0

TAB. 7.1 – Familles d'instances

Dans les sections suivantes des résultats sur chacun des résultats obtenus sont détaillés.

7.6.2 Comparaison des résultats

Les tables 7.2, 7.3, 7.4 et 7.5 sont construites comme suit. Pour chaque instance (quelques noms ont été raccourcis par souci de lisibilité), on indique le temps de détection des symétries (T), la validité (SAT), le nombre des symétries universelles ($\forall$), le nombre de symétries existentielles ($\exists$), le temps nécessité par chacun des solveurs pour résoudre les trois formules suivantes (originale $\mathcal{F}$, $\mathcal{F}^m$, $\mathcal{F}^n$) sont indiqués.

Comme on peut le prévoir, en cas d'amélioration majeure, celle ci est dûe principalement à la présence de cycles universels (e.g. `biu*` ou tt toilet*). Concernant les solveurs considérés, seul NCQUBE n'obtient pas les résultats escomptés. On pense que le fait que CUBE élimine les littéraux monotones pourraient être à l'origine. Il faut noter aussi que l'ordre imposé par le préfixe peut décroître l'impact de certaines symétries. En effet, sur certaines familles comme `sort*`, qui contiennent généralement une seule symétrie universelle, ordonner celle-ci suivant l'ordre partiel du préfixe place le premier cycle universel autour du 400^{ime} rang, ce qui ne permet pas d'utiliser efficacement les cycles universels. En plus, la taille du QSBP $\cup$ SBP est autour de 4000 clauses alors que la formule originale contient seulement 8000 clauses. Ce qui explique les mauvais résultats de la majorité des solveurs sur cette famille excepté SKIZZO.

Cependant pour la famille `biu*`, les cycles universels se positionnent principalement dans les premiers rangs, en plus nombreuses sont les symétries de la forme $(x_i, \neg x_i)$. Ce qui permet de supprimer du quantificateur plusieurs variables universelles. Ceci explique les bons résultats de SEMPROPou SKIZZO par exemple. Les instances `chain*` qui sont généralement faciles pour la plupart des solveurs, semblent être plus dures pour SQBF. Toutefois, supprimer les symétries en utilisant l'approche N-QSB permet à SQBF de toutes les résoudre.

7.7 Conclusion

Dans ce chapitre, deux nouvelles approches duales pour supprimer les symétries dans les instances QBF ont été présentées. Notre pré-traitement supprime les symétries d'une formule QBF et génère une nouvelle formule équivalente pour la validité qui ne contient pas de symétries. À travers des expérimentations, on a montré que sur de nombreuses instances QBF, l'élimination des symétries permet d'améliorer les performances des solveurs utilisés sur de nombreuses instances. L'approche N-QSB est une adaptation du fameux "symmetry breaking predicates" au cas QBF. L'approche M-QSB quant à elle supprime les sous-parties existentielles des symétries en ajoutant la sous-partie existentielle du SBP et celle correspondant à la sous-partie universelle est traduite sous forme de modèle et est ajoutée disjonctivement à la formule. Dans les choix expérimentaux on a opté, dans l'approche M-QSB, pour la transformation vers une forme conjonctive. Si l'approche M-QSB présente l'avantage de ne pas affecter l'ordre du préfixe, au contraire l'approche N-QSB introduit un nouvel ordre du préfixe. Un tel changement peut avoir des conséquences sur les choix heuristiques des solveurs. Pour les symétries elles-mêmes, on a opté pour l'ordre lexicographique car il garantit l'acyclicité du graphe de précédence. Cependant, on peut noter que l'ordre lexicographique ne constitue pas toujours le meilleur ordre. D'un autre côté, l'approche M-QSB sous sa forme CNF souffre du problème du littéral ajouté aux clauses de la formule originale. Les résultats expérimentaux montrent que la suppression des symétries dans les QBF améliore les performances des solveurs sur de nombreux problèmes.

Contrairement au cas SAT, où la majorité des instances symétriques sont fortement symétriques (e.g. chaines d'équivalences, problème des pigeons, etc.) le temps de calcul après suppression des symétries avoisine les quelques secondes. Dans le cas QBF, rares sont les instances fortement symétriques et l'ordre du préfixe peut avoir un grand impact sur l'efficacité de la symétrie. En effet, comme on peut le prévoir, on aimerait avoir plus des symétries sur des variables universelles plutôt que sur les variables existentielles dû au fait que les deux branches d'une variable universelle doivent être parcourues en général et le

Instance	pT	$\forall$	$\exists$	SAT	$\mathcal{F}$	N-QSB	M-QSB
ken.flash04.C-d2	1.82	0	1	N	–	**875.00**	**875.00**
ken.flash08.C-d2	1.90	0	1	Y	–	**1718.00**	**1718.00**
ken.flash06.C-d3	3.37	0	1	Y	4.93	**3.123**	**3.13**
ken.flash06.C-d4	6.23	0	1	N	52.78	**12.00**	**12.00**
sortnetsort7.AE.008	4.20	1	177	N	37.68	-	-
sortnetsort6.AE.003	0.13	1	49	Y	**0.45**	–	–
sortnetsort7.v.009	0.20	1	0	Y	–	**618.00**	–
sortnetsort8.v.003	0.05	1	0	N	100.77	42.00	**24.38**
sortnetsort6.v.004	0.02	1	0	N	244.00	**85.70**	171.00
lut4_AND_fXOR	0.04	3	8	N	370.00	**0.34**	1.10
toilet_c_10_01.19	0.08	2	0	Y	392.00	–	**124.00**
toilet_c_10_01.17	0.08	2	0	N	133.00	706.00	**22.00**
TOILET7.1.iv.13	0.03	2	0	N	995.00	196.00	**140.00**
TOILET6.1.iv.11	0.03	2	0	N	21.70	2.22	**2.12**
TOILET7.1.iv.14	0.03	2	0	Y	103.00	**3.17**	3.21
TOILET10.1.iv.20	0.13	2	0	Y	–	**0.94**	241.00
nusmv.tcas2.B-f2	4.76	0	7	N	959.00	**934.00**	**934.00**
nusmv.guidance2.C-f3	6.77	0	1	Y	310.00	**292.00**	**292.00**
cmu.gigamax.B-d3	0.90	2	3	Y	–	–	**417.00**
cmu.gigamax.B-f2	0.68	1	2	Y	201.00	**143.00**	196.00
k_ph_p-10	0.28	0	1	N	91.00	**34.18**	**34.18**
k_path_n-5	0.02	3	1	Y	12.00	**3.18**	19.92
C880.blif0.101.0001	0.12	0	14	N	–	**1054.00**	**1054.00**
C5315.blif_0.100.2000	2.24	2	35	Y	–	**0.09**	–
C5315.blif_0.100.2001	2.24	2	35	Y	–	**0.11**	–
C499.blif_0.101.0000	0.17	0	2	N	165.00	**89.00**	**89.00**
BLOCKS3i.5.3	0.05	0	1	N	1083.00	**307.00**	312.00

TAB. 7.2 – Résultats obtenus avec NCQUBE

Instance	pT	$\forall$	$\exists$	SAT	$\mathcal{F}$	N-QSB	M-QSB
biu.p010-OPF02-c08	12.87	48	43	Y	–	–	**1.26**
biu.p010-IPF05-c10	12.74	48	43	Y	–	–	**84.38**
sortnetsort5.AE.003	0.06	4	34	Y	–	571.00	**500.00**
sortnetsort9.v.012	1.00	1	0	Y	1616.00	56.83	805.00
sortnetsort9.v.011	0.85	1	0	Y	1229.00	**47.78**	1018.00
sortnetsort7.v.007	0.12	1	0	Y	102.00	35.63	**13.06**
lut4_AND_fXOR	0.04	3	8	N	10.10	**0.74**	8.66
toilet_c_10_01.19	0.08	2	0	N	338.00	142.00	**68.11**
toilet_c_10_01.18	0.07	2	0	N	97.87	75.16	**19.69**
toilet_c_10_01.17	0.06	2	0	N	26.45	21.01	**9.21**
S-edau-17	2.24	2	0	N	794.00	**14.22**	134.00
S-edau-15	2.17	2	0	N	–	–	**101.00**
S-adeu-43	2.30	2	0	N	680.00	**207.00**	900.00
S-adeu-18	2.39	2	0	N	190.00	**42.5**	712.00
ken.flash06.C-f4	6.37	0	3	N	1220.00	**93.72**	**93.72**
ken.flash06.C-f3	3.43	0	3	N	539.00	**52.73**	**52.73**
cmu.gigamax.B-f3	1.52	2	3	N	274.00	294.00	**249.00**
x20.19	0.03	0	2	Y	99.32	**85.39**	**85.00**
x25.14	0.04	0	2	Y	643.00	**355.00**	**355.00**
C880.blif_0.101.0001	0.10	0	14	Y	462.00	**440.00**	**440.00**
C5315.blif_0.101.0000	3.57	0	36	N	6.97	4.62	**4.59**
k_path_n-12	0.11	7	1	Y	10.57	**1.10**	10.27
k_path_p-10	0.05	3	1	N	11.16	**0.86**	2.76
k_path_p-13	0.11	7	1	N	9.85	**1.16**	10.32
k_path_p-16	0.11	3	1	N	11.11	9.73	**0.2**
k_branch_n-8_stricted	0.77	1	1	Y	–	824.93	–
k_branch_p-14	3.36	0	1	N	49.81	**43.31**	**43.31**
k_grz_n-17	0.05	0	8	Y	91.48	**71.73**	**71.73**
k_grz_n-18	0.06	0	11	Y	–	**105.00**	**105.00**
k_ph_n-17	4.67	0	1	Y	196.00	**174.00**	**174.00**
k_ph_n-18	6.65	0	1	Y	203.00	**188.00**	**174.00**
k_ph_p-11	0.48	0	1	N	1282.00	**1224.00**	**1224.00**
k_ph_n-19	7.90	0	1	Y	–	**138.00**	**138.00**

TAB. 7.3 – Résultats obtenus avec SKIZZO

Instance	pT	∀	∃	SAT	$\mathcal{F}$	N-QSB	M-QSB
lut4_2_f1	0.01	1	6	Y	–	**8.81**	–
lut4_2_fXOR	0.01	3	6	Y	172.00	**0.07**	8.95
lut4_AND_f1	0.02	0	7	Y	1200.00	**508.00**	**507.92**
lut4_3_fAND	0.12	1	10	Y	3.65	**1.45**	6.30
lut4_AND_fXOR	0.04	3	8	N	–	**16.83**	17.09
sortnetsort8.v.009	0.35	1	0	Y	1288.00	–	**169.00**
sortnetsort7.v.007	0.12	1	0	Y	339.00	975.00	**11.28**
sortnetsort6.v.004	0.02	1	0	N	–	**648.00**	–
sortnetsort7.v.003	0.03	1	0	N	562.00	**123.00**	136.00
CHAIN18v.19	1.07	18	0	Y	265.00	**0.11**	40.08
CHAIN19v.20	1.40	19	0	Y	593.00	**0.12**	–
CHAIN17v.18	0.82	17	0	Y	121.00	**0.09**	263.00
CHAIN20v.21	1.71	20	0	Y	1353.00	**0.13**	–
C5315.blif0.101.0000	3.57	0	36	N	–	**811.00**	**811.00**
C5315.blif0.101.0001	3.12	0	36	Y	–	**22.86**	**22.86**
k_path_p-6	0.02	3	1	N	159.00	**98.77**	450.00
k_branch_n-4_stricted	0.09	1	1	Y	179.00	**95.65**	323.00
k_ph_n-14	1.45	0	1	Y	–	**349.00**	**349.00**
k_grz_n-13	0.03	0	6	Y	142.00	**133.00**	**133.00**
k_grz_n-14	0.04	0	8	Y	215.00	**199.00**	**199.00**
TOILET7.1.iv.13	0.03	2	0	N	568.00	**18.50**	53.64
TOILET16.1.iv.32	1.28	4	0	Y	433.00	–	**0.53**
toilet_c_08_01.14	0.04	3	0	N	13.32	**2.97**	3.85
toilet_c_08_01.15	0.04	3	0	N	31.27	**11.06**	17.67
toilet_c_10_01.14	0.04	2	0	N	17.81	20.17	**2.23**
toilet_c_10_01.15	0.05	2	0	N	31.02	111.00	**7.01**
toilet_c_10_01.16	0.06	2	0	N	275.00	598.00	**33.01**
toilet_c_10_01.17	0.06	2	0	N	1642.00	–	**156.00**

TAB. 7.4 – Résultats obtenus avec SQBF

Instance	pT	$\forall$	$\exists$	SAT	$\mathcal{F}$	N-QSB	M-QSB
biu.p005-OPF03-c09	4.61	14	1	Y	–	47.79	**10.61**
biu.p010-IPF05-c07	6.56	27	1	Y	–	50.75	**13.14**
biu.p010-OPF02-c10	12.84	48	43	Y	–	26.93	**0.59**
lut4_3_fAND	0.12	1	10	Y	28.60	8.81	**0.35**
lut4_AND_fXOR	0.04	3	8	N	–	**1.63**	2.33
toilet_c_10_01.20	0.08	2	0	N	1468.00	5.00	**1.35**
toilet_c_10_01.19	0.08	2	0	N	1444.00	**0.40**	274.00
toilet_c_10_01.18	0.07	2	0	N	592.00	**0.30**	83.94
toilet_c_10_01.17	0.06	2	0	N	160.00	**0.25**	24.58
TOILET7.1.iv.13	0.03	2	0	N	75.42	14.03	**13.30**
TOILET7.1.iv.14	0.03	2	0	Y	**7.10**	11.54	11.93
sortnetsort5.AE.006	0.26	7	67	N	**396.00**	–	–
sortnetsort7.v.004	0.05	1	0	N	–	**1203.00**	–
C5315.blif_0.10_1.00_0_1	3.12	0	36	Y	–	**809.00**	**809.00**
k_path_n-18	0.22	9	1	Y	–	**887.00**	–
k_path_n-19	0.26	10	1	Y	613.00	**29.89**	–
k_path_p-13	0.11	7	1	N	640.00	**45.94**	379.00
k_path_p-20	0.26	10	1	N	–	**435.00**	–
k_branch_n-10	1.04	0	1	Y	1056.00	**994.00**	**994.00**
k_grz_n-13	0.03	0	6	Y	425.00	**362.00**	**362.00**
k_grz_p-13	0.03	0	4	N	431.00	**406.00**	**406.00**
k_branch_n-8_stricted	0.77	1	1	Y	57.71	52.30	54.816
k_branch_p-11	1.40	0	1	N	355.00	**345.00**	**345.00**
T-adeu-29	1.90	2	0	N	1.17	1.44	**1.04**
T-adeu-40	1.76	2	0	N	2.96	**1.41**	1.59
k_ph_n-10	0.25	0	1	Y	1052.00	**660.00**	**660.00**

TAB. 7.5 – Résultats obtenus avec SEMPROP

fait d'éliminer une branche universelle ne peut que bénéficier à la recherche. Cependant la pertinence des symétries universelles peut être mise en cause par l'ordre du préfixe. En effet, réordonner les symétries suivant l'ordre du préfixe peut renvoyer les cycles universels à la fin de la symétrie et la présence des cycles existentiels en début de la symétrie. Ceci décroît l'efficacité des cycles universels. Sur quelques instances il arrive que le premier cycle universel apparaît après des dizaines de cycles existentiels ce qui rend la propagation d'une variable universelle (requantifiée) peu probable.

7.8 Travaux futurs

Si les deux approches ont apporté des réponses à quelques questions essentielles et plus précisément à l'utilité des symétries dans le cas QBF, plusieurs voies restent à explorer. Tout d'abord sur le plan pratique, l'approche M-QSB a été testée sous sa forme conjonctive, reste à voir les performances de cette approche sous sa forme CNF/DNF. Ensuite, en cas de symétries nous voulons voir si la pondération de ces variables par une heuristique de choix de variables peut avoir un impact sur la recherche et sur la pertinence des symétries.

En ce qui concerne l'approche N-QSB, une question a été relevée et qui concerne la présence des out-littéraux positifs et négatifs. Des premières pistes de recherches ont été données pour traiter ce cas. Cependant une étude plus approfondie de la faisabilité et de la complexité du problème sont à explorer.

Finalement, une question naturelle concerne les symétries elles-mêmes. En effet, comme définies, les symétries d'une QBF $\mathcal{F} = \mathcal{Q}X\mathcal{M}$ sont les symétries qui laissent le préfixe invariant. Conséquence les symétries d'une QBF, comme définies, est un sous-ensemble des symétries de $\mathcal{M}$. Généraliser le mécanisme de suppression des symétries d'une QBF au cas de sa matrice n'est pas facile. En effet, soit la formule QBF suivante :

$$\mathcal{F} = \forall x_1 \exists y_1 (x_1 \vee y_1) \wedge (\neg x_1 \vee \neg y_1).$$

La formule $\mathcal{F}$ admet $\sigma = (x_1, y_1)$ comme symétrie. Le SBP correspondant est le suivant : $sbp(\sigma) = (\neg x_1 \vee y_1)$. Ce sbp est constitué d'une seule clause qui n'est pas totalement universelle. Cependant ajouter cette clause à $\mathcal{F}$ la rend non valide alors qu'elle est valide. Éudier comment utiliser les symétries de la matrice d'une QBF qui ne sont pas des symétries de la QBF est une perspective intéressante.

Conclusion générale

Dans cette thèse nous nous sommes intéressés à la résolution de deux problèmes fondamentaux à la fois sur le plan théorique mais aussi sur le volet applicatif : la satisfiabilité propositionnelle (SAT) et les formules booléennes quantifiées (QBF). Chacun des problèmes occupe une place de choix dans une classe de complexité importante : la classe des problèmes NP-Complets pour SAT et la classe des problèmes PSPACE-Complets pour QBF. Face à une telle complexité, les contributions apportées dans cette thèse visent à élargir un peu plus la classe des problèmes "traitables en pratique" et donc à renforcer la dimension applicative de ces deux problèmes.

Dans le cadre du problème SAT, nous avons d'abord formalisé et prouvé l'optimalité en terme de sauts du schéma classique d'analyse de conflits (CDCL) et plus particulièrement de la clause assertive générée suivant le principe du "First UIP". Ce résultat tranche avec la question du choix entre les différents UIPs. Il renforce aussi tout l'intérêt à proposer de nouvelles extensions pour espérer apprendre des clauses de meilleure qualité. C'est l'objet de notre seconde contribution : la proposition de deux nouveaux schémas d'apprentissage permettant d'étendre de manière originale les approches actuelles. Dans le premier, nous avons proposé une extension du graphe d'implication classique qui intègre de nouvelles clauses, appelées arcs inverses, habituellement ignorées par le schéma classique, car issues de la partie satisfiable de la formule. Nous avons également montré dans ce cadre comment on peut utiliser cette extension ou arcs inverses pour améliorer le niveau des retours-arrière, mais aussi pour réduire la taille de la clause assertive produite.

Le second schéma d'apprentissage est une nouvelle façon d'apprendre de nouvelles informations sur le processus de recherche en cours sans nécessairement attendre qu'un conflit soit rencontré. C'est pour cette raison que nous avons appelé ce nouveau schéma d'apprentissage "Learning from success". Ce schéma a pour but de découvrir de nouvelles implications pour les littéraux propagés. En d'autres termes, pour un littéral propagé de manière classique au niveau i, notre approche est capable d'apprendre de nouvelles raisons exprimant que ce littéral aurait du être propagé au niveau $j < i$. Nous avons ensuite proposé une première intégration de notre approche dans les solveurs SAT pour corriger les niveaux d'implications des littéraux et réarranger ainsi l'interprétation partielle courante. Ces nouvelles implications induisent par effet de bord, une amélioration de la qualité du graphe d'implication classique et donc de l'apprentissage à partir des conflits.

Notre troisième contribution a concerné la résolution parallèle de SAT. Contrairement aux approches existantes, généralement basées sur le principe de "diviser pour régner", dans la conception de notre solveur parallèle ManySAT, nous avons exploité les points faibles des solveurs SAT modernes : le manque de robustesse (ou stabilité) et la sensibilité aux réglages des paramètres. La méthode que nous avons proposée est basée sur un portfolio de stratégies complémentaires avec une politique de partage de clauses. Ces stratégies bien choisies avaient pour but de coopérer pour trouver une preuve plus courte. ManySAT est un solveur multicore, il utilise différentes stratégies de redémarrage, d'apprentissage, d'heuristiques pour les variables et les littéraux (choix de la polarité). On peut noter aussi qu'une autre contribution (mentionnée mais non developpée dans ce rapport) sur la politique de redémarrages est intégré dans ManySAT. Cette nouvelle politique dynamique se base sur des mesures pertinentes de la hauteur moyenne

des sauts-arrière et des arbres de recherche pour déterminer la valeur du “cutoff”. Le solveur ManySAT a été classé premier solveur parallèle (“gold medal”) à la dernière SAT-Race 2008 organisée en Chine.

La dernière contribution sur SAT a porté sur la proposition d’un nouveau cadre de représentation de formules CNF en graphe. La représentation graphe-SAT que nous avons proposée étend la représentation bien connue des formules 2-SAT en graphe. Cette nouvelle représentation est un graphe valué. Les nœuds représentent les littéraux et les arcs étiquetés par des ensemble de littéraux (contexte) représentent les clauses. Cette représentation capture mieux la structure des formules et des dépendances entre les littéraux. Dans ce cadre nous avons montré l’équivalence entre la résolution classique et la fermeture transitive du graphe. Pour montrer l’intérêt de cette nouvelle représentation, nous avons ensuite proposé et développé différentes utilisations ou applications possibles : le calcul d’ensembles “2-SAT strong backdoor”, la génération d’instances SAT difficiles et la simplification de formules CNF.

Pour la partie QBF, nous avons proposé deux approches pour supprimer les symétries dans les formules booléennes quantifiés. Notre but est de permettre aux solveurs de disposer de procédures indépendantes, capables de supprimer les symétries de la formule avant la résolution (en pré-traitement). Cette question est difficile et les approches proposées dans cette thèse sont les seules à l’heure actuelle capable d’éliminer les symétries par ajout de contraintes supplémentaires.

La première approche permet de transformer la QCNF en une formule CNF/DNF. La projection des symétries sur les quantificateurs existentiels est traduite en un SBP et est ajoutée conjonctivement à la formule et celle des universels est transformée en un ensemble de modèles ajouté disjonctivement à la formule.

La seconde approche est une extension du SBP classique au cas QBF. Cette extension a été motivée par le fait que le SBP utilisé dans le cas SAT ne peut être utilisé dans le cas QBF dans sa forme classique. Pour répondre aux contraintes posées par le préfixe et par la présence de variables universellement quantifiées, nous avons introduit, en plus du SBP classique, de nouvelles contraintes que nous avons appelées QSBP, et nous avons utilisé une représentation en graphe de précédence et un tri topologique pour déduire un nouveau préfixe à la formule QBF.

Perspectives

Ce travail ouvre de nombreuses perspectives pour de futurs travaux de recherche :

– *Solveurs SAT modernes* : Il s’agit sans conteste de proposer une formalisation en terme de stratégies de résolution de l’essence d’un solveur SAT moderne. Le succès des solveurs modernes est sans doute lié à une alchimie presque parfaite entre les composantes suivantes : heuristique, redémarrage et apprentissage. Toutefois, on pourra être content de voir que sur les instances industrielles, la majorité des instances sont résolues en moins de 15 minutes, mais déçu du fait que même si on fixe le temps limite de résolution à une ou deux heure de temps CPU, le nombre d’instances de plus que les solveurs modernes peuvent résoudre est vraiment médiocre. Un palier est donc atteint selon notre point de vue. Certaines des stratégies développées dans le cadre des solveurs SAT modernes doivent être repensées pour résoudre cette catégorie importante d’instances industrielles difficiles. La mise en place d’une nouvelle méthode de résolution plus constructive est un important challenge que nous souhaitons étudier.
  Les schémas d’apprentissage proposés constituent une première réponse à ce problème. Ces schémas au-delà de l’amélioration du niveau de retour-arrière ou de la réduction de la clause assertive constituent des façons “intelligentes” d’introduire de la résolution. Les voies ouvertes par le graphe d’implication étendu n’ont pas été épuisées et plusieurs pistes sont à explorer dans ce cadre au-delà des limitations posées par la recherche des arcs inverses.
– *Graphe-SAT* Enfin, dans la partie graphe, la représentation sous forme de graphe ouvre de nombreuse perspectives. Dans ce cadre, on veut voir dans la suite comment définir une distance entre

les nœuds, capable de donner une approximation des plus cours chemins (en terme de résolution) entre un litéral et son complémentaire. La réponse à cette question nous permettra d'avoir une approximation des variables du "backbone". Une importante direction de recherche concerne la mise en œuvre de nouvelles classes polynômiales en utilisant la nouvelle représentation en graphe-SAT.

– *Résolution parallèle de SAT et QBF* : Le solveur parallèle proposé constitue aussi une vraie réponse aux difficultés des solveurs modernes. Néanmoins plusieurs questions restent en suspens. Les résultats de la compétition Sat-Race 2008 ont montré, qu'un portfolio de solveurs bien choisi peut être meilleur que les approches parallèles majoritairement basées sur le principe de "diviser pour régner" mais aussi par rapport aux meilleurs solveurs SAT séquentiels. Il est clair que l'association en parallèle de deux stratégie performantes ne mène pas forcément à une stratégie performante dans le cadre du parallèle et vice versa. Alors comment rendre deux stratégies complémentaires et orthogonales à la fois, est une question difficile qui mérite d'être étudiée. De manière générale, mettre en œuvre un cadre formel pour la résolution parallèle de SAT est une perspective très importante pour nous. Travailler sur la résolution parallèle des QBF pourrait être une réponse aux challenges posés dans ce domaine. Dans ce cadre, nous souhaitons étendre et/ou utiliser la philosophie à l'origine de la conception de ManySAT au cas de parallèle QBF.
– *Learning dans QBF* : bien évidemment, une adaptation du nouveau cadre d'analyse de conflits (graphes d'implications étendus) au cas QBF ne peut que être bénéfique car l'existence de littéraux universels peut conduire à des gains plus substantiels.
– *Symétries en QBF* : nous avons proposé des réponses quant à la suppression de symétries en QBF. Mais plusieurs questions restent à étudier. La première concerne l'ordre des variables. On a opté dans l'implémentation pour des raisons justifiées pour l'ordre lexicographique, cependant cet ordre ne constitue pas l'ordre optimal pour une recherche. Trouver un "bon" ordre des variables n'est pas une question aisée. La question dépasse le cadre QBF. En effet, proposer le meilleur ordre pour l'élimination des symétries est une question centrale en SAT et CSP aussi. L'ordre utilisé influence directement les éventuels bénéfices que l'on peut tirer à supprimer les symétries. Deuxièmement, concernant les symétries elles-mêmes, nous avons défini une symétrie d'une QBF comme une symétrie qui laisse le quantificateur invariant, cependant il existe d'autres symétries entre les variables de quantificateurs différents qu'on appelle des symétries inter-quantificateurs. La question est comment supprimer ces symétries.

Bibliographie

[AKERS 1978] S. AKERS. « Binary Decision Diagrams ». *IEEE Transactions on Computers*, 27(6) :509–516, 1978.

[ALOUL *et al.* 2003a] Fadi A. ALOUL, Arathi RAMANI, Igor L. MARKOV, et Karem A. SAKALLAH. « Shatter : Efficient Breaking for Boolean Satisfiability ». Dans *proceedings of the international conference on Design Automation Conference (DAC)*, pages 836–839. ACM Press, 2003.

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

[ALOUL *et al.* 2006] Fadi A. ALOUL, Karem A. SAKALLAH, et Igor L. MARKOV. « Efficient Symmetry Breaking for Boolean Satisfiability ». *IEEE Trans. Computers*, 55(5) :549–558, 2006.

[ASPVALL *et al.* 1979] B. ASPVALL, M. PLASS, et R. TARJAN. « A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. ». *Information Processing Letters*, 8(3) :121–123, 1979.

[AUDEMARD & SAIS 2005] Gilles AUDEMARD et Lakhdar SAIS. « A Symbolic Search Based Approach for Quantified Boolean Formulas ». Dans *SAT*, pages 16–30, 2005.

[AUDEMARD *et al.* 2004] G. AUDEMARD, B. MAZURE, et L. SAIS. « Dealing with symmetries in Quantified Boolean Formulas ». Dans *proceedings of SAT*, pages 257–262, 2004.

[AUDEMARD *et al.* 2007a] Gilles AUDEMARD, Saïd JABBOUR, et Lakhdar SAIS. « Efficient Symmetry Breaking Predicates for Quantified Boolean Formulae ». Dans *proceedings of SymCon - Symmetry in Constraitns - CP workshop*, 2007.

[AUDEMARD *et al.* 2007b] Gilles AUDEMARD, Saïd JABBOUR, et Lakhdar SAIS. « Symmetry Breaking in Quantified Boolean Formulae ». Dans *proceedings of IJCAI*, pages 2262–2267, 2007.

[AUDEMARD *et al.* 2008a] Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Saïd JABBOUR, et Lakhdar SAIS. « A Generalized Framework for Conflict Analysis ». Dans *SAT*, pages 21–27, 2008.

[AUDEMARD *et al.* 2008b] Gilles AUDEMARD, Saïd JABBOUR, et Lakhdar SAIS. « SAT graph-based representation : A new perspective ». *J. Algorithms*, 63(1-3) :17–33, 2008.

[BACCHUS & WINTER 2003] F. BACCHUS et J. WINTER. « Effective Preprocessing with Hyper-Resolution and Equality Reduction. ». Dans *proceedings of SAT*, pages 341–355, 2003.

[BACCHUS 2002] Fahiem BACCHUS. « Enhancing Davis Putnam with Extended Binary Clause Reasoning ». Dans *Proceedings of AAAI*, pages 613–619, 2002.

[BAYARDO, JR. & SCHRAG 1997] Roberto J. BAYARDO, JR. et Robert C. SCHRAG. « Using CSP Look-Back Techniques to Solve Real-World SAT Instances ». Dans *Proceedings of AAAI*, pages 203–208, 1997.

[BEAME *et al.* 2004] Paul BEAME, Henry A. KAUTZ, et Ashish SABHARWAL. « Towards Understanding and Harnessing the Potential of Clause Learning ». *J. Artif. Intell. Res. (JAIR)*, 22 :319–351, 2004.

[BENEDETTI 2004] Marco BENEDETTI. « Evaluating QBFs via Symbolic Skolemization ». Dans *LPAR*, pages 285–300, 2004.

[BENEDETTI 2005a] M. BENEDETTI. « sKizzo : a Suite to Evaluate and Certify QBFs ». Dans *proceedings of CADE*, pages 369–376, 2005.

[BENEDETTI 2005b] Marco BENEDETTI. « Extracting Certificates from Quantified Boolean Formulas ». Dans *IJCAI*, pages 47–53, 2005.

[BENHAMOU & SAÏDI 2007] Belaïd BENHAMOU et Mohamed Réda SAÏDI. « Local Symmetry Breaking During Search in CSPs ». Dans *proceedings of CP*, volume 4741 de *LNCS*, pages 195–209, 2007.

[BENHAMOU & SAIS 1994] Belaid BENHAMOU et Lakhdar SAIS. « Tractability through Symmetries in Propositional Calculus ». *Journal of Automated Reasoning*, 12(1) :89–102, février 1994.

[BENNACEUR 1995] Hachemi BENNACEUR. « Boolean approch for representing and solving constraint-satisfaction problems ». Dans *AI*IA*, pages 163–174, 1995.

[BIERE 2004] Armin BIERE. « Resolve and Expand ». Dans *SAT*, 2004.

[BLOCHINGER *et al.* 2003] W. BLOCHINGER, C. SINZ, et W. KÜCHLIN. « Parallel propositional satisfiability checking with distributed dynamic learning ». *Parallel Computing*, 29(7) :969–994, 2003.

[BÖHM & SPECKENMEYER 1996] Max BÖHM et Ewald SPECKENMEYER. « A Fast Parallel SAT-Solver - Efficient Workload Balancing ». *Ann. Math. Artif. Intell.*, 17(3-4) :381–400, 1996.

[BOOLE 1854] G. BOOLE. *Les lois de la pensée*. Mathesis, 1854.

[BOROS *et al.* 1994] E. BOROS, P.L. HAMMER, et X. SUN. « Recognition q-Horn formulae in linear time ». *Journal of Discrete Applied Mathematics*, 55 :1–13, 1994.

[BOUFKHAD 1996] Yacine BOUFKHAD. « *Aspects probabilistes et algorithmiques du problème de satisfiabilité* ». phd thesis, Université de Paris 6, Laboratoire d' Informatique de Paris 6 (LIP6), décembre 1996.

[BRAFMAN 2001] R. BRAFMAN. « A Simplifier for Propositional Formulas with Many Binary Clauses ». Dans *proceddings of IJCAI*, pages 515–522, 2001.

[BRAUNSTEIN *et al.* 2005] A. BRAUNSTEIN, M. MÉZARD, et R. ZECCHINA. « Survey propagation : An algorithm for satisfiability. ». *Random Struct. Algorithms*, 27(2) :201–226, 2005.

[BRISOUX *et al.* 1999] Laure BRISOUX, Éric GRÉGOIRE, et Lakhdar SAIS. « Improving Backtrack Search for SAT by Means of Redundancy ». Dans *ISMIS*, pages 301–309, 1999.

[BRUYNOOGHE 1980] M. BRUYNOOGHE. « Analysis of Dependencies to Improve the Behaviour of Logic Programs ». Dans *5th Conference on Automated Deduction, CADE'5*, pages 293–305, 1980.

[BUBECK & BÜNING 2007] Uwe BUBECK et Hans Kleine BÜNING. « Bounded Universal Expansion for Preprocessing QBF ». Dans *SAT*, pages 244–257, 2007.

[BÜNING *et al.* 1995] Hans Kleine BÜNING, Marek KARPINSKI, et Andreas FLÖGEL. « Resolution for Quantified Boolean Formulas ». *Inf. Comput.*, 117(1) :12–18, 1995.

[BUNO & BUNING 1993] M. BUNO et H.K. BUNING. « Report on SAT competition ». *Bulletin of the European Association for Theoritical Computer Science*, 49 :143–151, Feb. 1993.

[CADOLI *et al.* 1998] Marco CADOLI, Andrea GIOVANARDI, et Marco SCHAERF. « An Algorithm to Evaluate Quantified Boolean Formulae ». Dans *Proceedings of AAAI*, pages 262–267, 1998.

[C.E. BLAIR & LOWE 1986] R.G. Jeroslow C.E. BLAIR et J.K. LOWE. « Some results and experiments in programming techniques for propositional logic ». *Comput. Oper. Res.*, 13(5) :633–645, 1986.

[CHATALIC & SIMON 2000] Philippe CHATALIC et Laurent SIMON. « Multi-resolution on compressed sets of clauses ». Dans *ICTAI*, pages 2–10, 2000.

[CHATALIC & SIMON 2001] Philippe CHATALIC et Laurent SIMON. « Multiresolution for SAT Checking ». *International Journal on Artificial Intelligence Tools*, 10(4) :451–481, 2001.

[CHRABAKH & WOLSKI 2003] Wahid CHRABAKH et Rich WOLSKI. « GrADSAT : A Parallel SAT Solver for the Grid ». Rapport technique, UCSB Computer Science Technical Report Number 2003-05, 2003.

[CHU & STUCKEY 2008] G. CHU et P. J. STUCKEY. « Pminisat : a parallelization of minisat 2.0 ». Rapport technique, Sat-race 2008, solver description, 2008.

[COHEN *et al.* 2005] David A. COHEN, Peter JEAVONS, Christopher JEFFERSON, Karen E. PETRIE, et Barbara M. SMITH. « Symmetry Definitions for Constraint Satisfaction Problems ». Dans *proceedings of CP*, pages 17–31, 2005.

[COOK 1971] S. A. COOK. « The complexity of theorem-proving procedures ». Dans *Proceedings of the Third Annual ACM Symposium on*

Theory of Computing, pages 151–158, New York (USA), 1971. Association for Computing Machinery.

[COSTE-MARQUIS *et al.* 2006] Sylvie COSTE-MARQUIS, Hélène FARGIER, Jérôme LANG, Daniel Le BERRE, et Pierre MARQUIS. « Representing Policies for Quantified Boolean Formulae ». Dans *proceedings of KR*, pages 286–297, 2006.

[CRAWFORD *et al.* 1996] J. CRAWFORD, M. GINSBERG, E. LUCK, et A. ROY. Symmetry-Breaking Predicates for Search Problems. Dans *proceedings of KR*, pages 148–159. 1996.

[CRAWFORD 1992] James CRAWFORD. « A theorical Analysis of Reasoning by Symmetry in First Order Logic ». Dans *Proceedings of Workshop on Tractable Reasonning, AAAI92*, pages 17–22, juillet 1992.

[DARRAS *et al.* 2005] Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, Bertrand MAZURE, Richard OSTROWSKI, et Lakhdar SAIS. « Using Boolean Constraint Propagation for Sub-clauses Deduction ». Dans *CP*, pages 757–761, 2005.

[DARWICHE 2004] A. DARWICHE. « New Advances in Compiling CNF to Decomposable Negational Normal Form ». Dans *Proceedings of ECAI*, pages 328–332, 2004.

[DAVIS & H. 1960] Martin DAVIS et Putnam H.. « A computing Procedure for quantification Theory ». *Commun. ACM*, 7(7) :201–215, 1960.

[DAVIS *et al.* 1962] M. DAVIS, G. LOGEMANN, et D. W. LOVELAND. « A machine program for theorem-proving. ». *Communications of the ACM*, 5(7) :394–397, 1962.

[DECHTER 1989] R. DECHTER. « Enhancement Schemes for Constraint Processing : Backjumping, Learning, and cutset Decomposition ». *Artificial Intelligence*, 41 :273–312, 1989.

[DEQUEN & DUBOIS 2006] Gilles DEQUEN et Olivier DUBOIS. « An Efficient Approach to Solving Random -satProblems ». *J. Autom. Reasoning*, 37(4) :261–276, 2006.

[DIXON & GINSBERG 2002] H. DIXON et M. GINSBERG. « Inference Methods for a Pseudo-Boolean Satisfiability Solver. ». Dans *proceedings of AAAI*, pages 635–640, 2002.

[DUBOIS *et al.* 1996] Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD, et Jacques CARLIER. « SAT versus UNSAT ». Dans D.S. JOHNSON et M.A. TRICK, éditeurs, *Second DIMACS Challenge, 1993*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, pages 415–436, 1996.

[EÉN & BIERE 2005] N. EÉN et A. BIERE. « Effective Preprocessing in SAT Through Variable and Clause Elimination. ». Dans *proceedings of SAT*, pages 61–75, 2005.

[EÉN & SÖRENSSON 2003] Niklas EÉN et Niklas SÖRENSSON. « An Extensible SAT-solver ». Dans *SAT*, pages 502–518, 2003.

[FORMAN & SEGRE 2002] Sean L. FORMAN et Alberto M. SEGRE. « Nagsat : A randomized, complete, parallel solver for 3-sat. SAT2002 ». Dans *Proceedings of SAT2002*, pages 236–243, 2002.

[FOX & LONG 1999] Maria FOX et Derek LONG. « The Detection and Exploitation of Symmetry in Planning Problems ». Dans *proceedings of IJCAI*, pages 956–961, 1999.

[GALLO & URBANI 1989] Giorgio Gallo GALLO et Giampaolo URBANI. « Algorithms for Testing the Satisfiability of Propositional Formulae. ». 7(1) :45–61, juillet 1989.

[GENT *et al.* 2005] Ian P. GENT, Tom KELSEY, Steve LINTON, Iain MCDONALD, Ian MIGUEL, et Barbara M. SMITH. « Conditional Symmetry Breaking ». Dans *proceedings of CP*, pages 256–270, 2005.

[GINSBERG 1993] Matthew L. GINSBERG. « Dynamic Backtracking ». *Journal of Artificial Intelligence Research (JAIR)*, 1 :25–46, 1993.

[GIUNCHIGLIA *et al.* 2001a] E. GIUNCHIGLIA, M. NARIZZANO, et A. TACCHELLA. « Quantified Boolean Formulas satisfiability library (QBFLIB) », 2001. `http://www.qbflib.org`.

[GIUNCHIGLIA *et al.* 2001b] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « An Analysis of Backjumping and Trivial Truth in Quantified Boolean Formulas Satisfiability ». Dans *AI*IA*, pages 111–122, 2001.

[GIUNCHIGLIA *et al.* 2001c] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Backjumping for Quantified Boolean Logic Satisfiability ». Dans *IJCAI*, pages 275–281, 2001.

[GIUNCHIGLIA *et al.* 2001d] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « QuBE », 2001.
http ://www.mrg.dist.unige.it/˜qube.

[GIUNCHIGLIA *et al.* 2002] E. GIUNCHIGLIA, M. MARATEA, et A. TACCHELLA. « Dependent and independent variables in propositional satisfiability ». Dans *proceedings of ECLAI*, pages 296–307, 2002.

[GIUNCHIGLIA *et al.* 2004] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Monotone Literals and Learning in QBF Reasoning ». Dans *CP*, pages 260–273, 2004.

[GIUNCHIGLIA *et al.* 2006] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas ». *Journal of Artificial Intelligence Research*, 26 :371–416, 2006.

[GOLDBERG & NOVIKOV 2002] E. GOLDBERG et Y. NOVIKOV. « BerkMin : A Fast and Robust SAT-Solver ». Dans *Proceedings of International Conference on Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, 2002.

[GOMES *et al.* 1997] Carla P. GOMES, Bart SELMAN, et Nuno CRATO. « Heavy-Tailed Distributions in Combinatorial Search ». Dans *Principles and Practice of Constraint Programming*, pages 121–135, 1997.

[GOMES *et al.* 1998] Carla P. GOMES, Bart SELMAN, et Henry KAUTZ. « Boosting Combinatorial Search Through Randomization ». Dans *Proceedings of AAAI*, pages 431–437, Madison, Wisconsin, 1998.

[GOMES *et al.* 2000] C. P. GOMES, B. SELMAN, N. CRATO, et H. KAUTZ. « Heavy-tail phenomena in satisfiability and constraint satisfaction ». *Journal of Automated Reasoning*, pages 67 – 100, 2000.

[GREGOIRE *et al.* 2005] E. GREGOIRE, B. MAZURE, R. OSTROWSKI, et L. SAIS. « Automatic extraction of functional dependencies ». Dans *proceedings of SAT*, volume 3542 de *LNCS*, pages 122–132, 2005.

[HUANG 2007] Jinbo HUANG. « The Effect of Restarts on the Efficiency of Clause Learning ». Dans *IJCAI*, pages 2318–2323, 2007.

[ICHI MINATO 1996] Shin ichi MINATO. « Fast factorization method for implicit cube set representation ». *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(4) :377–384, 1996.

[IWAMA 1989] Kazuo IWAMA. « CNF-satisfiability test by counting and polynomial average time ». *SIAM Journal on Computing*, 18(2) :385–391, avril 1989.

[JEROSLOW & WANG 1990] Robert G. JEROSLOW et Jinchang WANG. « Solving Propositional Satisfiability Problems ». *Ann. Math. Artif. Intell.*, 1 :167–187, 1990.

[JR. & SCHRAG 1997] Roberto J. Bayardo JR. et Robert SCHRAG. « Using CSP Look-Back Techniques to Solve Real-World SAT Instances ». Dans *AAAI/IAAI*, pages 203–208, 1997.

[JURKOWIAK *et al.* 2001] Bernard JURKOWIAK, Chu Min LI, et Gil UTARD. « Parallelizing Satz Using Dynamic Workload Balancing ». Dans *In Proceedings of Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, pages 205–211, 2001.

[KAUTZ & SELMAN 1997] H. KAUTZ et D. McAllester B. SELMAN. « Exploiting variable Dependency in Local Search ». Dans *Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya (Japan), 1997.

[KAUTZ *et al.* 1997] H. KAUTZ, D. MCALLESTER, et B. SELMAN. « Exploiting variable Dependency in Local Search ». Dans *"Abstracts of the Poster Sessions of IJCAI-97"*, 1997.

[KAUTZ *et al.* 2002] H. KAUTZ, E. HORVITZ, Y. RUAN, C. GOMES, et B. SELMAN. « Dynamic restart policies ». Dans *Proceedings of AAAI*, pages 674–682, 2002.

[KRISHNAMURTHY 1985] Balakrishnan KRISHNAMURTHY. « Shorts Proofs for Tricky Formulas ». *Acta Informatica*, 22 :253–275, 1985.

[LARDEUX *et al.* 2002] Frédéric LARDEUX, Frédéric SAUBION, et Jin-Kao HAO. « A hybrid Genetic Algorithm for the Satisfiability Problem ». Dans *Proc. of the 1rst International Workshop on Heuristics*, Beijing, China, jul 2002.

[LARRABEE 1992] Tracy LARRABEE. « Test pattern generation using Boolean satisfiability ». *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(1) :4–15, 1992.

[LETZ 2002a] R. LETZ. « Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas ». Dans *Proceedings of Tableaux*, pages 160–175, 2002.

[LETZ 2002b] Reinhold LETZ. « Semprop », 2002. http ://www4.informatik.tu-muenchen.de/~letz/semprop.

[LEWIS *et al.* 2007] M. LEWIS, T. SCHUBERT, et B. BECKER. « Multithreaded SAT Solving ». Dans *12th Asia and South Pacific Design Automation Conference*, 2007.

[LI & ANBULAGAN 1997] Chu Min LI et ANBULAGAN. « Heuristics Based on Unit Propagation for Satisfiability Problems ». Dans *IJCAI (1)*, pages 366–371, 1997.

[LI 2003] C. LI. « Equivalent literal propagation in the DLL procedure. ». *Discrete Applied Mathematics*, 130(2) :251–276, 2003.

[LU *et al.* 2003] F. LU, L. WANG, K. CHENG, et R. HUANG. « A Circuit SAT Solver With Signal Correlation Guided Learning. ». Dans *proceedings of DATE*, pages 892–897, 2003.

[LUBY *et al.* 1993a] Michael LUBY, Alistair SINCLAIR, et David ZUCKERMAN. « Optimal Speedup of Las Vegas Algorithms ». Dans *ISTCS*, pages 128–133, 1993.

[LUBY *et al.* 1993b] Michael LUBY, Alistair SINCLAIR, et David ZUCKERMAN. « Optimal speedup of Las Vegas algorithms ». *Information Processing Letters*, 47 :173–180, 1993.

[LYNCE & MARQUES-SILVA 2004] I. LYNCE et J. MARQUES-SILVA. « Hidden Structure in Unsatisfiable Random 3-SAT : an Empirical Study ». Dans *proceedings of ICTAI*, pages 246–251, 2004.

[LYNCE *et al.* 2001] Inês LYNCE, Luís BAPTISTA, et João P. Marques SILVA. « Towards Provably Complete Stochastic Search Algorithms for Satisfiability ». Dans *EPIA*, pages 363–370, 2001.

[MAAREN & NORDEN 2005] H. Van MAAREN et L. Van NORDEN. « Correlations between Horn fractions, satisfiability and solver performance for fixed density random 3-CNF instances ». *Annals of Mathematics and Artificial Intelligence*, 44 :157– 177, 2005.

[MARQUES-SILVA & SAKALLAH 1996] Joao P. MARQUES-SILVA et Karem A. SAKALLAH. « GRASP - A New Search Algorithm for Satisfiability ». Dans *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[MCALLESTER 1980] D.A. MCALLESTER. « An Outlook on Truth Maintenance ». AI Memo551, August 1980. MIT AI Laboratory.

[MCKAY 1990] B. MCKAY. « nauty user's guide (version 1.5) ». Rapport technique, 1990.

[MOSKEWICZ *et al.* 2001] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG, et Sharad MALIK. « Chaff : Engineering an Efficient SAT Solver ». Dans *Proceedings of DAC*, pages 530–535, 2001.

[NADEL *et al.* 2006] Alexandre NADEL, Maon GORDON, Amit PATI, et Ziad HANA. « Eureka-2006 SAT solver ». Dans *In solver description SAT-race*, 2006.

[NARIZZANO *et al.* 2006] M. NARIZZANO, L. PULINA, et A. TACCCHELLA. « QBF solvers competitive evaluation (QBFEVAL) », 2006. http ://www.qbflib.org/qbfeval.

[PAN & VARDI 2004] Guoqiang PAN et Moshe Y. VARDI. « Symbolic Decision Procedures for QBF ». Dans *CP*, pages 453–467, 2004.

[PARIS *et al.* 2006] Lionel PARIS, Richard OSTROWSKI, Pierre SIEGEL, et Lakhdar SAIS. « Computing Horn Strong Backdoor Sets Thanks to Local Search ». Dans *ICTAI*, pages 139–143, 2006.

[PIETTE *et al.* 2008] CÃ©dric PIETTE, Youssef HAMADI, et Sais LAKHDAR. « Vivifying propositional clausal formulae ». Dans *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 525–529, Patras (Greece), July 2008.

[PIPATSRISAWAT & DARWICHE 2007a] Knot PIPATSRISAWAT et Adnan DARWICHE. « A Lightweight Component Caching Scheme for Satisfiability Solvers ». Dans *SAT*, pages 294–299, 2007.

[PIPATSRISAWAT & DARWICHE 2007b] Knot PIPATSRISAWAT et Adnan DARWICHE. « RSat 2.0 : SAT Solver Description ». Rapport technique D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

[RAMANI *et al.* 2006] Arathi RAMANI, Igor L. MARKOV, Karem A. SAKALLAH, et Fadi A. ALOUL. « Breaking Instance-Independent Symmetries In Exact Graph Coloring ». *Journal of Artificial Intelligence Research (JAIR)*, 26 :289–322, 2006.

[RINTANEN 1999] Jussi RINTANEN. « Improvements to the Evaluation of Quantified Boolean Formulae ». Dans *IJCAI*, pages 1192–1197, 1999.

[RINTANEN 2001] Jussi RINTANEN. « Partial Implicit Unfolding in the Davis-Putnam Procedure for Quantified Boolean Formulae ». Dans *LPAR*, pages 362–376, 2001.

[RISH & DECHTER 2000] Irina RISH et Rina DECHTER. « Resolution versus Search : Two Strategies for SAT ». *J. Autom. Reasoning*, 24(1/2) :225–275, 2000.

[ROBINSON 1965] John Alan ROBINSON. « A Machine-Oriented Logic Based on the Resolution Principle ». *J. ACM*, 12(1) :23–41, 1965.

[RYVCHIN & STRICHMAN 2008] Vadim RYVCHIN et Ofer STRICHMAN. « Local Restarts ». Dans *SAT*, pages 271–276, 2008.

[SABHARWAL *et al.* 2006] Ashish SABHARWAL, Carlos ANSÓTEGUI, Carla P. GOMES, Justin W. HART, et Bart SELMAN. « QBF Modeling : Exploi-

ting Player Symmetry for Simplicity and Efficiency ». Dans *proceedings of SAT*, pages 382–395, 2006.

[SAIS 2008] Lakhdar SAIS. *Problème SAT : Progrès et Défis*. Hermes Publishing Ltd, 2008.

[SAMULOWITZ & BACCHUS 2005] Horst SAMULOWITZ et Fahiem BACCHUS. « Using SAT in QBF ». Dans *proceedings of CP*, pages 578–592, 2005.

[SAMULOWITZ & BACCHUS 2006] Horst SAMULOWITZ et Fahiem BACCHUS. « Binary Clause Reasoning in QBF ». Dans *SAT*, pages 353–367, 2006.

[SANG *et al.* 2005] Tian SANG, Paul BEAME, et Henry A. KAUTZ. « Heuristics for Fast Exact Model Counting ». Dans *SAT*, pages 226–240, 2005.

[SCHIEX & VERFAILLIE 1993] T. SCHIEX et G. VERFAILLIE. « Nogood Recording for Static and Dynamic Constraint Satisfaction Problems ». Dans *International Conference on Tools with Artificial Intelligence (ICTAI'93)*, pages 48–55, 1993.

[SELMAN *et al.* 1992] Bart SELMAN, Hector J. LEVESQUE, et David MITCHELL. « GSAT : A New Method for Solving Hard Satisfiability Problems ». pages 440–446, 1992.

[SELMAN *et al.* 1997] Bart SELMAN, Henry A. KAUTZ, et David A. MCALLESTER. « Ten Challenges in Propositional Reasoning and Search ». Dans *IJCAI (1)*, pages 50–54, 1997.

[SHANNON 1938] C.E. SHANNON. « A symbolic analysis of relay and switching circuits ». *Transactions AIEE*, 57 :713–719, 1938.

[SILVA & SAKALLAH 1996] João P. Marques SILVA et Karem A. SAKALLAH. « Conflict Analysis in Search Algorithms for Satisfiability ». Dans *ICTAI*, pages 467–469, 1996.

[SINZ & DIERINGER 2005] C. SINZ et E. DIERINGER. « DPvis - A Tool to Visualize the Structure of SAT Instances. ». Dans *proceedings of SAT*, pages 257–268, 2005.

[SKOLEM 1928] T. SKOLEM. « Über die mathematische logik ». 1928.

[STALLMAN & SUSSMAN 1977] R.M. STALLMAN et G.J. SUSSMAN. « Forward Reasoning and Dependency-Directed backtraking in a System for Computer AIded Circuit Analysis ». *Artificial Intelligence*, 9 :135–196, October 1977.

[STÉPHAN 2006] Igor STÉPHAN. « Boolean Propagation Based on Literals for Quantified Boolean Formulae ». Dans *proceedings of ECAI*, pages 452–456, 2006.

[SUBBARAYAN & PRADHAN 2004] Sathiamoorthy SUBBARAYAN et Dhiraj K. PRADHAN. « NiVER : Non Increasing Variable Elimination Resolution for Preprocessing SAT instances ». Dans *SAT*, 2004.

[THIFFAULT *et al.* 2004] C. THIFFAULT, F. BACCHUS, et T. WALSH. « Solving Non-clausal Formulas with DPLL search ». Dans *proceedings of CP*, pages 663–678, 2004.

[TSEITIN 1968] G.S. TSEITIN. « On the complexity of derivations in the propositional calculus ». Dans H.A.O. SLESENKO, éditeur, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[WALSH 2006] Toby WALSH. « General Symmetry Breaking Constraints ». Dans *proceedings of CP*, pages 650–664, 2006.

[WILLIAMS *et al.* 2003] R. WILLIAMS, C. GOMES, et B. SELMAN. « Backdoors to Typical Case Complexity ». Dans *proceedings of IJCAI*, pages 1173–1178, 2003.

[ZHANG & MALIK 2002a] L. ZHANG et S. MALIK. « Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation ». Dans *Proceedings of the CP*, pages 200–215, 2002.

[ZHANG & MALIK 2002b] Lintao ZHANG et Sharad MALIK. « Quaffle », 2002. http ://ee.princeton.edu/˜chaff/Quaffle.php.

[ZHANG & STICKEL 1994] H. ZHANG et M. E. STICKEL. « Implementing the Davis-Putnam algorithm by tries ». Rapport technique, Artificial Intelligence Center, SRI International, Menlo, 1994.

[ZHANG *et al.* 1996] H. ZHANG, M. P. BONACINA, et J. HSIANG. « PSATO : a Distributed Propositional Prover and its Application to Quasigroup Problems ». *Journal of Symbolic Computation*, 21 :543–560, 1996.

[ZHANG *et al.* 2001] Lintao ZHANG, Conor F. MADIGAN, Matthew W. MOSKEWICZ, et Sharad MALIK. « Efficient Conflict Driven Learning in Boolean Satisfiability Solver ». Dans *ICCAD*, pages 279–285, 2001.

[ZHANG 1997] H. ZHANG. « SATO : An Efficient Propositional Prover ». Dans *International Conference on Automated Deduction, CADE'97*, 1997.

[ZHANG 2006] Lintao ZHANG. « Solving QBF with Combined Conjunctive and Disjunctive Normal Form ». Dans *proceedings of AAAI*, 2006.

[ZHENG 2005] Yaling ZHENG. « Survey of Techniques for Detecting and Exploiting Symmetry in Constraint Satisfaction Problems ». Rapport technique, Constraint Systems Laboratory, University of Nebraska, 2005.

Résumé

Cette thèse traite de deux problèmes combinatoires difficiles : la satisfiabilité propositionnelle - SAT (NP-Complet) et la résolution de formules booléennes quantifiées - QBF (PSPACE Complet). Dans le cas SAT, nous présentons tout d'abord deux extensions efficaces du schéma classique d'analyse des conflits, communément appelé CDCL "Conflict Driven Clause Learning". En intégrant de nouveaux arcs (ou implications) au graphe d'implication classique, cette première extension permet d'améliorer la hauteur des sauts lors du retour-arrière. La seconde extension permet de déduire de nouvelles raisons pour les littéraux impliqués et permettre ainsi un réarrangement de l'interprétation partielle courante. Ensuite un nouveau solveur parallèle ManySAT est décrit. ManySAT utilise un portfolio complémentaire d'algorithmes séquentiels conçus grâce à une minutieuse variation dans les principales stratégies de résolution. Ces stratégies sont combinées au partage de clauses dans le but d'améliorer les performances globales. Enfin, une nouvelle représentation sous forme de graphe d'une formule CNF est présentée. Elle étend le graphe d'implication utilisé dans le cas du fragment polynomial 2-SAT en associant des valuations (ensembles de littéraux, appelés conditions) aux arcs. La résolution classique est reformulée en utilisant la fermeture transitive du graphe. Trois utilisations concrètes de cette nouvelle représentation sont présentées. La première concerne le calcul d'ensembles 2-SAT "strong backdoors". Dans la seconde, nous exploitons cette représentation pour la génération d'instances SAT difficiles. Alors que dans la troisième, nous dérivons une nouvelle technique de prétraitement des formules CNF.

Dans le cas QBF, nous présentons deux approches duales pour supprimer les symétries. La première consiste à reformuler les interprétations symétriques en pré-modèles qu'on peut ajouter préalablement à la QBF en plus du SBP ("Symetry Breaking Predicates") restreint aux cycles existentiels permettant d'obtenir une formule mixte CNF/DNF. La deuxième approche est une extension du SBP classique de SAT vers QBF. Elle est réalisée grâce à une transformation du préfixe de la formule et à l'ajout (en plus du SBP classique) de nouvelles contraintes appelées QSBP ("Quantified Symetry Breaking Predicates").

Mots-clés: Satisfiabilité (SAT), Formules Boolénnes Quantifiées (QBF), Symétries, Résolution Parallèle.

Abstract

In this thesis we deal with two hard combinatorial problems : Propositional Satisfiability - SAT (NP-Complete) and Quantified Boolean formulae - QBF (PSPACE complete). In SAT, we first present two extensions of conflict analysis scheme obtained thanks to additional clauses generally ignored in classical implication graph. In the first extension, an extended implication graph is proposed and used to improve the backjumping level of the classical asserting clauses. The second extension allows us to extract new and strong reasons for the implication of a given literal. An original approach for correcting the implication levels of some propagated literals is then derived. After, we present ManySAT a parallel SAT solver. The design of ManySAT benefits from the main weaknesses of modern SAT solvers, their sensitivity to parameter tuning and their lack of robustness. ManySAT uses a portfolio of complementary sequential algorithms obtained thanks to a careful design or variation in the main solver strategies. These orthogonal strategies combined with clause sharing aims to improve further the ManySAT overall performance. Finally, a new graph representation of CNF formula is presented. This representation extend the classical 2-SAT implication graph with additional labelled edges, called conditional implications. Classical resolution is then reformulated using the transitive closure of the graph. Three applications are described. The first one concern the computation of 2-SAT strong backdoor sets. In the second, we exploit this graph representation to generate hard SAT instances while in the third one we derive a new and efficient pre-processing technique for CNF formulae.

In the QBF context, we present two dual approaches to break symmetries. The first approach concerns the transformation of the QBF to a new satisfiability equivalent QBF formula in CNF/DNF form. The DNF contains models, whereas the CNF represent the constraints obtained by the projection of the SBP on the existential quantifiers. The second approach is an adaptation of the classical SAT SBP to the QBF case. In this approach, a new prefix is generated and additional constraints, called QSBP "quantified symmetry breaking predicates", are conjunctively added to the classical SBP.

Keywords: Satisfiability (SAT), Quantified Boolean Formulas (QBF), Symmetry, Parallel Resolution