UNIVERSITÉ D'ARTOIS

Résolution de problèmes autour de SAT

Habilitation à Diriger des Recherches

(Spécialité Informatique)

Université d'Artois

présentée et soutenue publiquement le 29 novembre 2010

par

Gilles AUDEMARD

Composition du jury

Rapporteurs :	Arnaud LALLOUET	Professeur à l'Université de Caen
	Frédéric SAUBION	Professeur à l'Université d'Angers
	Thomas SCHIEX	Directeur de Recherche à l'INRA
Examinateurs :	Chu Min LI	Professeur à l'Université de Picardie
	Lakhdar SAÏS	Professeur à l'Université d'Artois
Directeur :	Christophe LECOUTRE	Professeur à l'Université d'Artois

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

Mis en page avec la classe thloria.

Remerciements

Voici une petite page pour remercier toutes les personnes qui, de près ou de loin, m'ont aidé, soutenu et encouragé toutes ces années.

Mes remerciements vont en tout premier lieu à Arnaud LALLOUET, Frédéric SAUBION et Thomas SCHIEX qui, malgré des emplois du temps bien chargés, ont trouvé du temps pour rapporter ce document et siéger dans mon jury. Un grand merci également à Chu Min LI pour avoir accepté de participer à mon jury.

Un grand merci à Lakhdar SAÏS pour avoir siégé à ce jury, mais aussi pour avoir grandement contribué à mon intégration dans le nord, et avoir accepté de travailler avec moi : beaucoup de travaux présentés ici ont été réalisé en sa compagnie.

Un autre grand merci à Christophe LECOUTRE pour avoir accepté de diriger cette habilitation, mais aussi pour les très bons moments passés en sa compagnie (je me souviendrai longtemps de cette soirée où nous avons écouté les chansons les plus ringardes possibles).

Les remerciements à Christophe et Lakhdar m'amènent tout naturellement à passer au CRIL et à l'IUT. Un immense merci à vous tous pour l'accueil que vous m'avez réservé, mais également pour la joie et la bonne humeur qui règnent dans ces lieux et permettent de travailler en totale sérénité.

L'intégration de ma famille dans le nord a été facilitée, non pas par le soleil, mais par les nombreuses personnes qui, au fil des années, nous ont témoignées leur amitié, un grand merci à vous tous, vous vous reconnaitrez.

Lors de l'écriture de ma thèse, je terminai par remercier toute ma famille, je réitère ces remerciements aujourd'hui : merci à vous tous et spécialement à Steph, Sally (je n'oublie pas la spéciale dédicace à Céline !), Theo, sans oublier le petit dernier : Tom.

Table des matières

Table des illustrations

Liste des figures

Liste des tables

Liste des algorithmes

Introduction générale

CE DOCUMENT PRÉSENTE l'ensemble des travaux réalisé depuis la fin de ma thèse. Ils ont été effectués au sein de l'IRST (Intituto per la Ricerca Scientifica e Tecnologica, Trento Italie) pour une petite partie et au sein du CRIL (Centre de Recherche en Informatique de Lens) pour la plus grande. Ces travaux sont le fruit d'une étroite collaboration avec des chercheurs de l'IRST, du CRIL, Lakhdar SAÏS et Bertrand MAZURE en tête. Je n'oublie pas Saïd JABBOUR et Jean-Marie LAGNIEZ, les doctorants que j'ai (eu) le plaisir de co-encadrer, mais également Laurent SIMON du LRI (Laboratoire de Recherche en Informatique). Les travaux présentés ici sont donc en partie les leurs.

Ces travaux ont un socle commun, le problème SAT (pour satisfaisabilité). J'ai commencé à travailler sur ce problème, et essentiellement sur l'algorithmique associée, lors de ma thèse. La dichotomie entre la simplicité à le formuler et la difficulté à le résoudre m'a toujours fasciné. Les recherches sur le problème SAT ont aujourd'hui cinquante ans. Depuis les premières procédures proposées pour résoudre ce problème [DP60, DLL62] jusqu'aux démonstrateurs actuels [MMZ+01, ES04], le chemin parcouru a été très important. Les recherches d'un niveau théorique ont permis de mettre en évidence aussi bien des classes de problèmes polynomiales [APT79] que difficiles pour la résolution [Hak85]. La mise en évidence des backdoors [WGS03], backbones [DD01] et de propriétés structurelles ont amélioré la compréhension du problème SAT et ont participé au développement de méthodes de plus en plus efficaces. Les progrès sont tels qu'aujourd'hui il n'est pas rare d'utiliser une transformation vers SAT pour résoudre des problèmes venant de domaines les plus variés (planification [KS92], vérification formelle [Bie09], diagnostic [SVV04]...). L'importance actuelle du problème SAT en recherche informatique est d'ailleurs soulignée par Edmund CLARKE, prix Turing 2007 : « *Clearly, efficient* SAT *solving is a key technology for 21st century computer science* » [BHvMW09].

C'est donc autour de la résolution du problème SAT que se situent une majorité de mes recherches. J'essaie de mieux comprendre les démonstrateurs de type CDCL (« *Conflict Driven, Clause Learning* ») qui ont un comportement assez imprévisible et dont certains mécanismes sont toujours mal appréhendés. Il me semble en effet que l'amélioration de ces démonstrateurs passera en partie par là. Au travers d'une ANR blanche, je travaille également sur les méthodes de recherche incomplètes pour prouver l'incohérence de certaines formules. En effet, de telles méthodes existent mais ne savent, dans la plupart des cas, que prouver la satisfaisabilité d'une formule. Enfin, au travers de nouvelles formes de représentation, j'essaie également de mieux cerner le problème SAT pour en déduire des nouvelles propriétés et autres techniques de pré-traitements.

Le manque d'expressivité de la logique propositionnelle peut être considéré comme un atout, puisque cela permet de coder naturellement en SAT des problèmes extrêmement variés mais également d'implanter relativement facilement des démonstrateurs SAT. Mais cela peut également être vu comme un inconvénient majeur à cause de la taille des formules générées mais aussi par l'impossibilité de coder certains problèmes comme ceux issus de la planification dans l'incertain [Rin99a], les logiques modales [PV03]. Des extensions du problème SAT autorisent une expressivité plus importante. C'est le cas du problème QBF (Quantified Boolean Formulae) où certaines variables peuvent être quantifiées universellement. Ce simple ajout n'est pas sans conséquence puisqu'il permet de représenter certains problèmes de manière plus concise que dans SAT et de coder également les problèmes cités plus haut. Mais, l'ajout du quantificateur universel place le problème QBF plus haut dans la hiérarchie de la complexité [SM73]. De mon coté, j'ai travaillé sur le problème QBF en orientant mes recherches sur deux axes différents. Tout d'abord, j'ai étudié la suppression des symétries pouvant exister dans certaines instances. J'ai également proposé un nouveau type de démonstrateur QBF qui s'abstrait de l'ordre imposé par les quantificateurs.

La satisfiabilité modulo théories (SMT) est une autre extension de SAT qui offre un degré d'expressivité élevée. Un problème SMT contient des atomes propositionnels classiques, mais également des atomes issus d'une théorie donnée. J'ai travaillé sur SMT en me focalisant sur les théories des équations linéaires. Les clauses d'un tel problème peuvent donc avoir la forme $(\neg x \vee (v_1 - v_2 > 3) \vee (v_2 - v_3 = 2))$. J'ai participé au développement d'un démonstrateur permettant de résoudre de telles formules, mais également à une application directe de l'expressivité offerte par SMT en généralisant la vérification formelle aux automates temporels.

Cet mémoire résume donc mes différentes activités de recherche. Il se compose de trois parties. La première d'entre elle est un synthèse des travaux de recherche. Elle est séparée en trois principaux chapitres. Le premier résume mes travaux autour de SAT, le second autour de QBF et enfin le troisième autour de SMT. Afin d'aider à la compréhension, cette synthèse est accompagnée d'une introduction aux travaux en relation avec mes recherches. La seconde partie de ce mémoire est un curriculum vitae présentant mes diverses activités de recherche, d'enseignement, mais également la liste de mes publications. Enfin, la troisième partie est une sélection de mes publications couvrant les divers domaines abordés dans la première partie.

Partie I

Synthèse des travaux de recherche

Sommaire

Chapitre

1

Autour de SAT

NOUS FÊTONS CETTE ANNÉE le cinquantième anniversaire de la procédure de DAVIS et PUTNAM (DP) [DP60] qui est l'origine des travaux portant sur le problème SAT. Depuis cette date, la recherche sur SAT a toujours été un domaine très actif. Des travaux d'ordre théorique ont permis une meilleure compréhension de ce problème et des travaux d'ordre plus pratique ont permis de mettre au point des démonstrateurs de plus en plus efficaces. Cela a d'ailleurs été le cas dès le départ, puisque en 1962 une nouvelle méthode améliorant la procédure DP était proposée [DLL62]. Aujourd'hui, il est courant d'utiliser un codage vers SAT pour résoudre des problèmes venant de domaines aussi variés que la planification [KS92] la vérification formelle [Bie09], la bio-informatique [LMS06], la cryptographie [MM00, SNC09]... Cette façon de faire, donne de très bons résultats dans la pratique, quelquefois meilleurs que des démonstrateurs ad-hoc. Cela est assez étrange, puisque seulement vingt ans en arrière, le problème SAT, premier problème à avoir été démontré NP-complet [Coo71], était surtout utilisé pour montrer que des problèmes étaient également NP-complets et donc intraitable dans le cas général [GJ79]. Une des raisons qui ont fait de SAT un problème beaucoup étudié est sa simplicité. En effet, la logique propositionnelle est la manière la plus simple de représenter des connaissances. Le codage sous forme SAT est donc relativement intuitif à imaginer.

Ce chapitre est donc dédié au problème SAT. La première section introduit les définitions nécessaires à la compréhension du reste du chapitre. Les travaux que nous avons effectués sur le problème SAT se situent principalement sur 3 axes. Dans le premier, nous nous sommes attelés à mieux comprendre et à améliorer les solveurs de type CDCL (*Conflict Driven, Clause Learning*). C'est l'objet de la section B. Nous avons également travaillé sur la recherche locale dédiée à la preuve d'insatisfaisabilité des problèmes et c'est l'objet de la section C. Enfin, nous avons effectué des recherches sur la représentation et le codage des formules propositionelles (voir la section D). Cette synthèse est accompagnée d'une introduction aux travaux en relation avec nos recherches.

A Pré-requis

Nous commençons cette section par introduire les différentes notions associées au problème SAT.

Définition 1.1
- Une *variable* propositionelle x peut prendre comme valeur de vérité *vrai* (encore noté 1 ou $\top$), ou *faux* (encore noté 0 ou $\bot$).
- Un *littéral* l est une variable x ou sa négation $\neg x$.
- Le littéral *opposé* (ou *complémentaire*) à l est noté $\neg l$.

Définition 1.2
- Une *clause* c est une disjonction finie de littéraux $c = l_1 \vee l_2 \dots l_m$. Elle pourra être représentée comme l'ensemble des littéraux $\{l_1, l_2, \dots, l_m\}$.
- Une clause *unitaire* contient un seul littéral.
- Une clause *binaire* contient deux littéraux.
- La taille d'une clause c, notée $|c|$, est le nombre de ses littéraux.
- Une clause est dite de *Horn* si elle contient au plus un littéral positif.
- Une clause est *tautologique* si elle contient un littéral et son opposé.
- Une clause c_1 *sous-somme* (ou *subsume*) une clause c_2 si $c_1 \subseteq c_2$.

Définition 1.3 Une formule Σ sous forme normale conjonctive (CNF) est une conjonction finie de clauses. Comme pour les clauses, une CNF pourra être indifféremment représentée par l'ensemble des clause qu'elle contient.

Définition 1.4 Soit une formule Σ, on note $\mathcal{L}(\Sigma)$ l'ensemble des littéraux qui apparaissent dans cette formule. De plus $\mathcal{L}(\Sigma) = \mathcal{L}^+(\Sigma) \cup \mathcal{L}^-(\Sigma)$ où $\mathcal{L}^+$ et L^- représentent respectivement les littéraux positifs et négatifs de Σ.

Définition 1.5
- Une *interprétation* $\mathcal{I}$ sur un ensemble $\mathcal{X}$ de variables propositionnelles est une fonction qui associe à chaque variable de $\mathcal{X}$ la valeur de vérité *vrai* ou *faux*. On pourra considérer une interprétation comme l'ensemble des littéraux vrais dans celle-ci.
- Une interprétation $\mathcal{I}$ satisfait un littéral l, si $l \in \mathcal{I}$. Dans le cas contraire, l'interprétation falsifie le littéral l.
- Étant donnée une formule CNF Σ, une interprétation $\mathcal{I}$ est *complète* (notée $\mathcal{I}_c$) si elle donne une valeur de vérité à toutes les variables de Σ. Autrement, elle est dite *partielle* (notée $\mathcal{I}_p$).
- Une interprétation $\mathcal{I}$ *satisfait* une clause c si elle satisfait au moins un littéral de c.
- Une interprétation $\mathcal{I}$ *falsifie* une clause c si tous les littéraux de c sont faux dans $\mathcal{I}$.
- Une clause c est *unisatisfaite* par une interprétation $\mathcal{I}$ si un seul littéral de c est vrai dans $\mathcal{I}$ et tous les autres littéraux sont *faux*.
- Une interprétation $\mathcal{I}$ satisfait une formule Σ si elle satisfait toutes les clauses de Σ. On dit alors que $\mathcal{I}$ est un *modèle* de Σ.
- Une interprétation partielle $\mathcal{I}_p$ est cohérente si elle ne falsifie pas de clauses (ce n'est pas nécessairement un modèle).
- Étant donnée une formule Σ et une interprétation $\mathcal{I}$ on note $\Sigma_{|\mathcal{I}}$ la formule Σ simplifiée par $\mathcal{I}$ (les clauses contenant des littéraux de $\mathcal{I}$ sont supprimées (car satisfaites) et les opposés des littéraux de $\mathcal{I}$ sont supprimés (car falsifiés)).

Définition 1.6 Une formule CNF Σ est *satisfaisable* (on dit aussi cohérente) si elle admet au moins un modèle. Dans le cas contraire elle est dite *insatisfaisable* (ou encore incohérente).

On peut maintenant définir le problème SAT :

Définition 1.7 Le problème SAT est le problème de décision qui consiste à déterminer si une formule propositionelle admet ou non un modèle.

Exemple 1.8
Soit la formule CNF $\Sigma = (x_1 \vee x_2) \wedge (\neg x_3) \wedge (x_3 \vee \neg x_1)$. Elle est satisfaisable et $\mathcal{I} = \{\neg x_1, x_2, \neg x_3\}$ est un modèle de Σ.

Historiquement, le problème SAT a été le premier problème à être démontré NP-complet [Coo71]. Nous ne donnons pas de détails sur la théorie de la NP-complétude dans ce mémoire ; pour de plus amples informations le lecteur se référera à [GJ79]. Néanmoins, il est important de noter qu'un problème NP-complet est un problème pour lequel il est facile (réalisable en temps polynomial) de montrer qu'une solution en est bien une, mais pour lequel il est difficile de trouver une telle solution. Ainsi, à l'heure actuelle, on ne connaît que des algorithmes en temps exponentiel en la taille des entrées pour résoudre les problèmes NP-complets. L'algorithmique autour de SAT s'efforce donc d'élargir autant que possible l'ensemble des instances que l'on sait résoudre. Malgré tout, il existe certaines classes d'instances qui s'avèrent être de complexité polynomiale. Elles ont un intérêt aussi bien théorique (la découverte de nouvelles classes polynomiales aide à mieux cerner le problème SAT), que pratique (exploitation durant la recherche). Nous donnons dans la définition ci-dessous quelques classes de formules pour lesquelles le problème SAT est de complexité polynomiale :

Définition 1.9 (Classes polynomiales pour SAT)
- Le problème 2-SAT : toutes les clauses ont exactement deux littéraux [APT79] (voir section 2)
- Le problème Horn-SAT : toutes les clauses sont des clauses de Horn [DG84]
- Le problème Horn-Renommable-SAT : problème se transformant en problème de Horn par renommage de certaines de ses variables [Lew78]

Une fois les classes polynomiales introduites, nous pouvons définir les notions de *backdoors* [WGS03]. Cela passe tout d'abord par la définition de sous-solveur :

Définition 1.10 (Sous-solveur [WGS03]) Étant donné une formule CNF Σ, A est un sous-solveur de Σ si il vérifie les conditions suivantes :
- Soit A résout Σ en déterminant si elle est ou non satisfaisable, soit il la rejète.
- A est de complexité polynomiale.
- Si A sait résoudre Σ alors pour tout littéral l, A sait résoudre $\Sigma_{|\{l\}}$.

Définition 1.11 (backdoor [WGS03]) Un ensemble non vide de variables $\mathcal{V}$ est un *backdoor* d'une formule Σ pour un sous-solveur A, si il existe une interprétation $\mathcal{I}_\mathcal{V}$ des variables de $\mathcal{V}$ telle que A résout $\Sigma_{|\mathcal{I}_\mathcal{V}}$.

Définition 1.12 (strong backdoor [WGS03]) Un ensemble non vide de variables $\mathcal{V}$ est un *strong backdoor* d'une formule Σ pour un sous-solveur A, si pour toute interprétation $\mathcal{I}_\mathcal{V}$ des variables de $\mathcal{V}$, A résout $\Sigma_{|\mathcal{I}_\mathcal{V}}$.

La notion de *strong backdoor* détermine la difficulté des problèmes. En effet, l'affectation des backdoors rend le problème résultant polynomial. La taille de l'ensemble strong backdoor est donc la complexité intrinsèque de la formule. Plus celui-ci est petit, plus la formule est "*simple*" à résoudre. Malheureusement, déterminer le plus petit ensemble strong backdoor d'une formule est un problème NP-difficile [Sze05]. À partir de là, de nombreux travaux ont été proposés pour calculer les plus petits ensembles backdoors possibles [GMOS05, POSS06]. Dans la section D, nous proposons deux algorithmes de recherche d'ensembles *strong backdoors*.

La résolution est la technique de base dans le raisonnement autour de SAT. Elle est utilisée dans l'algorithme de DAVIS et PUTNAM [DP60], ou encore pour simplifier les formules avant la recherche [LA97, EB05]. Elle est également l'opération de base utilisée dans l'analyse des conflits des solveurs CDCL (voir section 1.2).

Définition 1.13 Deux clauses c_1 et c_2 se résolvent si et seulement si il existe un littéral l tel que $l \in c_1$ et $\neg l \in c_2$. La clause $(c_1 - \{l\}) \cup (c_2 - \{\neg l\})$ est appelée résolvante en l de c_1 et c_2. Elle est notée $c_1 \otimes_l c_2$.

Exemple 1.14
Soient les clauses $c_1 = x_1 \vee x_2 \vee \neg x_3$ et $c_2 = \neg x_1 \vee x_2 \vee x_4$. On a alors $c_1 \otimes_{x_1} c_2 = x_2 \vee \neg x_3 \vee x_4$

La nouvelle clause $c_1 \otimes_l c_2$ ainsi obtenue est une conséquence logique des deux clauses initiales c_1 et c_2 : toute interprétation qui satisfait (resp. falsifie) $c_1 \otimes_l c_2$, satisfait (resp. falsifie) également c_1 et c_2. Cette clause peut donc être ajoutée à la formule initiale sans en changer sa nature.

B Solveurs CDCL

L'histoire des solveurs SAT commence en 1960 avec la procédure de DAVIS et PUTNAM [DP60]. L'algorithme qu'ils proposent consiste, à chaque étape d'un processus itératif, à supprimer une variable en effectuant toutes les résolutions possibles sur celle-ci. La notion de clause unitaire est également présente dans cet algorithme. Néanmoins, sa complexité spatiale est très élevée et il est quelque peu tombé en désuétude aujourd'hui (on notera quand même quelques travaux relatifs récents [DR94, CS01]). C'est en 1962 qu'apparaît la fameuse procédure de DAVIS, LOGEMAN et LOVELAND [DLL62]. Elle est directement issue des travaux de 1960, mais l'exploration se fait cette fois en réduisant la formule à chaque étape. C'est sur la base de cette procédure que de nombreux solveurs ont été proposés durant près de 40 ans [1]. Nombreuses sont les améliorations qui y ont été apportées : étude des heuristiques de décision [JW90, Fre95, DD01], étude des structures de données [ZS96], dépendances fonctionnelles [OGMS02], filtrage [ABS00, LB01]...

À la fin du vingtième siècle, deux publications révolutionnent le monde des solveurs SAT. Dans l'article « *Bounded Model Checking without BDDs* » [BCCZ99], les auteurs proposent d'utiliser des démonstrateurs SAT pour faire de la vérification formelle (*model checking*) de circuits électroniques [2]. Leurs travaux ont grandement amélioré les anciennes méthodes basées sur les BDD (voir la section 2.1 pour une introduction sur les BDD). Néanmoins, la taille des formules codant ces problèmes est importante et a donc fait apparaître de nouveaux besoins pour les démonstrateurs.

1. Suivant les articles, cette dernière procédure est nommée DP, DLL, DPLL, la dernière n'oubliant aucun des auteurs historiques !

2. Le prix Turing 2007 (équivalent des prix Nobels dans le domaine informatique) remporté par CLARKE, EMERSON et SIFAKIS montre l'importance de la vérification formelle en informatique.

Le second article au cœur de cette révolution est « *Chaff : Engeneering an efficient SAT solver* » [MMZ+01]. Les auteurs ont proposé un nouveau type de solveurs, les solveurs CDCL (« *Conflict Driven, Clause Learning* ») ou encore solveurs modernes (même si leur existence remonte déjà à 10 ans). Pour cela, ils ont utilisé des techniques déjà émergentes à l'époque (apprentissage et backjumping [MSS99, MSS96], redémarrages [GSK98]...) [3]. Ils ont introduit leur solveur comme une réécriture des solveurs DPLL classiques. Ils supposaient, entre autre, que la gestion du cache mémoire était une des grandes raisons de l'efficacité de leur démonstrateur [ZM03]. Aujourd'hui, la photographie est quelque peu différente et les solveurs CDCL sont plus vus comme des algorithmes de production de clauses [Hua07a]. De plus, nous y reviendrons dans la section 4.1, il est aujourd'hui prouvé que les solveurs de type CDCL sont plus puissants que les solveurs DPLL classiques.

Ces deux travaux ont donc radicalement changé la recherche autour de SAT. On est passé de l'utilisation de SAT pour montrer que des problèmes sont NP-complets et donc difficiles, à l'utilisation de SAT pour résoudre des problèmes difficiles.

Nous présentons dans le reste de cette section les solveurs de type CDCL ainsi que les différents mécanismes qui en font leur succès. A savoir, les structures de données paresseuses, l'analyse de conflit, l'apprentissage, le saut arrière, l'heuristique adaptative, et pour finir les redémarrages.

1 Architecture

L'algorithme 1.1 donne le schéma général d'une méthode de type CDCL. Une branche typique d'un tel solveur est une séquence de décisions suivies de propagations, répétée jusqu'à ce qu'un conflit survienne. Chaque littéral de décision (lignes 19–21) est affecté à un niveau de décision (d), les littéraux qui se déduisent (par propagation unitaire [4]) de cette décision ont le même niveau ($l@d$ indique que le littéral l est affecté au niveau d). Une interprétation $\mathcal{I}$ peut donc être écrite $\mathcal{I} = \{\langle \ell_{k_{1_1}} @1, \dots \ell_{k_{1_i}} @1\rangle, \langle \ell_{k_{2_1}} @2, \dots \ell_{k_{2_j}} @2\rangle \dots \langle \ell_{k_{d_1}} @d, \dots \ell_{k_{d_t}} @d\rangle\}$. Les $\langle$ et $\rangle$ partitionnent l'interprétation $\mathcal{I}$ suivant les différents niveaux de décision et les variables $\ell_{k_{1_1}}, \ell_{k_{2_1}}, \dots$ sont les variables de décision de chaque niveau. Si tous les littéraux sont affectés, alors $\mathcal{I}$ est un modèle de Σ (lignes 17–18). À chaque fois qu'un conflit est atteint par propagation unitaire (c est alors la clause falsifiée, lignes 6–7), un nogood γ est calculé (ligne 9) en utilisant une méthode donnée, le plus souvent le premier UIP (Unique Implication Point) [ZMMM01] et un niveau de backjump bl est calculé. À ce moment, on peut avoir prouvé l'incohérence de la formule. Si ce n'est pas le cas, on fait un saut arrière (on enlève de l'interprétation $\mathcal{I}$ des séquences de littéraux) et le nouveau niveau de décision devient égal au niveau de backjump (lignes 14–15). Enfin, de temps à autre, les solveurs CDCL forcent des redémarrages et dans ce cas, on remonte tout en haut de l'arbre de recherche (ligne 13). Nous détaillons ces diverses procédures ci-dessous.

1.1 Structures de données paresseuses

Lors de la propagation unitaire, il est nécessaire de connaître les clauses unisatisfaites. Jusqu'aux années 2000, les matrices creuses étaient utilisées. Pour chaque littéral, on stockait l'ensemble des clauses où il apparaissait. Avec l'apprentissage, cette structure de données est trop gourmande en mémoire. De plus, c'est durant l'opération de propagation unitaire que se concentre la majorité du temps de calcul d'un solveur SAT. A partir de ces constatations et améliorant des travaux déjà existants [ZS96], les auteurs de

3. Le prix CAV 2009 attribué aux auteurs de CHAFF [MMZ+01] et de GRASP [MSS96] montre l'impact des solveurs SAT au sein de la communauté de la vérification formelle.

4. L'opération de propagation unitaire consiste à forcer l'affectation à *vrai* d'un littéral qui se trouve dans une clause où tous les littéraux sauf ce dernier sont falsifiés. Cette clause devient donc unisatisfaite. Cette opération se répète jusqu'à saturation.

ALGORITHME 1.1 : Solveur de type CDCL

Input : Σ une formule CNF
Output : SAT ou UNSAT

```
1  I_p = ∅ ;                                              /* interprétation */
2  d = 0 ;                                                /* niveau de décision */
3  nb_c = 0 ;                                             /* numéro du conflit */
4  Γ = ∅ ;                                                /* ensemble des clauses apprises */
5  while (true) do
6      c = propagationUnitaire(Σ ∪ Γ,I_p);
7      if (c ≠null) then
8          nb_c = nb_c + 1 ;                              /* nouveau conflit */
9          γ =analyseConflit(Σ ∪ Γ,I_p,c);
10         bl = calculRetourArriere(γ,I_p);
11         if (bl < 0) then return UNSAT;
12         Γ = Γ ∪ {γ};
13         if (redémarrage(nb_c)) then bl = 0;
14         retourArriere(Σ ∪ Γ,I,bl);                     /* Mise à jour de I_p */
15         d = bl;
16     else
17         if (toutes les variables sont affectées) then
18             return SAT;
19         ℓ = choixLiteralDecision(Σ ∪ Γ);
20         d = d + 1;
21         I_p = I_p ∪ {⟨ℓ@d⟩};
22
23 end
```

CHAFF [MMZ+01] ont proposé une nouvelle structure de données, dite paresseuse. Nous expliquons la variante proposée dans [ES04].

La figure 1.1 schématise le fonctionnement de cette structure de données. Chaque clause possède deux littéraux, les deux premiers, témoignant qu'elle est au minimum binaire ou alors satisfaite («*watched literals*»). Ces deux témoins doivent donc être des littéraux non affectés à faux. De plus, chaque littéral connaît les clauses dont il est le témoin. Lorsque l'on affecte le littéral a_1 à faux, il ne peut plus être considéré comme témoin. On parcourt donc les clauses qu'il surveille. On cherche alors un autre littéral comme témoin (ici le littéral a_3). On note donc que lorsque un littéral devient faux mais qu'il n'est pas témoin de la clause, aucun calcul supplémentaire n'est nécessaire (littéral a_4). On continue comme cela (le littéral a_2 devient faux et a_5 devient témoin de la clause). Lorsque a_5 est affecté à faux, il n'y a plus de témoin possible. Deux cas se présentent alors :

- Soit l'autre témoin (ici a_3) est non affecté et dans ce cas, il devient un littéral unitaire.
- Soit il est également faux et la clause est alors falsifiée, un conflit est repéré durant la propagation unitaire. La clause est alors retournée pour pouvoir analyser ce conflit.

Lorsque l'on parcourt la clause à la recherche d'un littéral témoin, on peut découvrir que cette clause est déjà satisfaite et dans ce cas on va l'ignorer (elle ne peut pas devenir unisatisfaite). Le parcours des clauses satisfaites provoque donc un surcoût inutile. Des améliorations ont donc été proposées afin d'éviter ce surcoût : on rajoute à la structure de données un littéral (appelé littéral bloqué [NO05]),

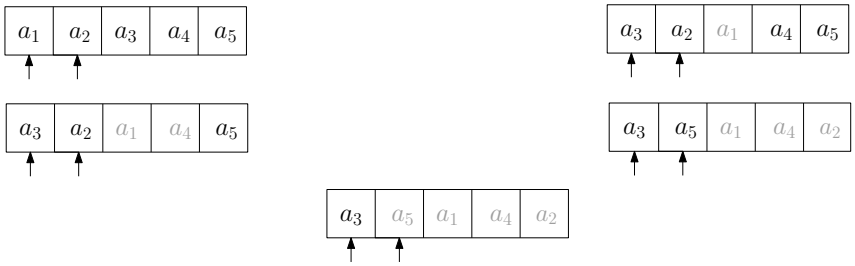


FIGURE 1.1 : *Comportement des structures de données paresseuses*. La clause est composée de 5 littéraux (les littéraux grisés sont falsifiés). L'ordre d'affectation des littéraux est a_1, a_4, a_2, a_5 (de gauche à droite et de haut en bas). a_3 devenant unitaire à la dernière étape.

celui-ci témoignant que la clause est satisfaite.

Remarque 1.15

- Il n'est pas nécessaire de mettre à jour les listes témoins lors d'un retour arrière, celui-ci n'impose donc aucun coût.
- On ne peut connaître l'état d'une clause (satisfaite, binaire...) qu'en la parcourant.
- La propagation d'un littéral n'impose de connaître et de visiter que les clauses que l'opposé de ce littéral surveille (gain en temps et en espace).

Reste ensuite à gérer les listes des clauses regardées par les littéraux. De nombreuses possibilités existent, visant pour la plupart d'entre elles à minimiser les problèmes de cache [CHS09, Bie08a]. Notez également que les clauses binaires de la formule sont gérées à part. En effet, les littéraux témoins sont inutiles dans ce cas et la connaissance des clauses binaires est importante pour les solveurs SAT.

1.2 Analyse des conflits et retour arrière

Les premiers travaux sur les sauts arrières [Pro93] et l'apprentissage [Dec90] ont été réalisé sur les CSP (problèmes de satisfaction de contraintes). Les interactions entre le monde des CSP et celui de SAT ont permis l'élaboration et le développement de ces techniques au problème SAT [MSS96, BJS97], qui, depuis, sont devenues incontournables.

Le but de l'analyse de conflit est de trouver un ensemble de littéraux responsables d'un conflit, i.e. d'une clause falsifiée. Au moyen d'une nouvelle clause (un « *nogood* » ou encore *clause assertive*) obtenue par résolution, cette analyse de conflit indique au solveur qu'il n'existe pas de solution dans un certain espace de recherche et propose un nouvel espace de recherche à visiter pour la suite. Ce nogood va alors être ajouté à la formule initiale. Dès lors, le solveur ne pourra plus être confronté au même échec. Le schéma d'analyse de conflits le plus couramment utilisé est le UIP «Unique Implication Point» [ZMMM01]. Il est intimement lié à la notion de graphe d'implication. Nous présentons ces différents concepts à l'aide d'un exemple.

Considérons la formule CNF suivante :

$c_1 = x_1 \vee x_2$ $\quad c_4 = x_3 \vee x_5 \vee x_6$ $\quad c_7 = \neg x_9 \vee x_{10} \vee x_{11}$ $\quad c_{10} = x_{12} \vee x_{14}$
$c_2 = \neg x_2 \vee \neg x_3$ $\quad c_5 = \neg x_6 \vee x_7 \vee x_8$ $\quad c_8 = x_8 \vee \neg x_{11} \vee \neg x_{12}$ $\quad c_{11} = x_{12} \vee \neg x_6 \vee x_{15}$
$c_3 = \neg x_2 \vee \neg x_4 \vee \neg x_5$ $\quad c_6 = \neg x_4 \vee x_8 \vee x_9$ $\quad c_9 = x_{12} \vee \neg x_{13}$ $\quad c_{12} = x_{13} \vee \neg x_{14} \vee \neg x_{16}$
$c_{13} = \neg x_{14} \vee x_{15} \vee x_{16}$

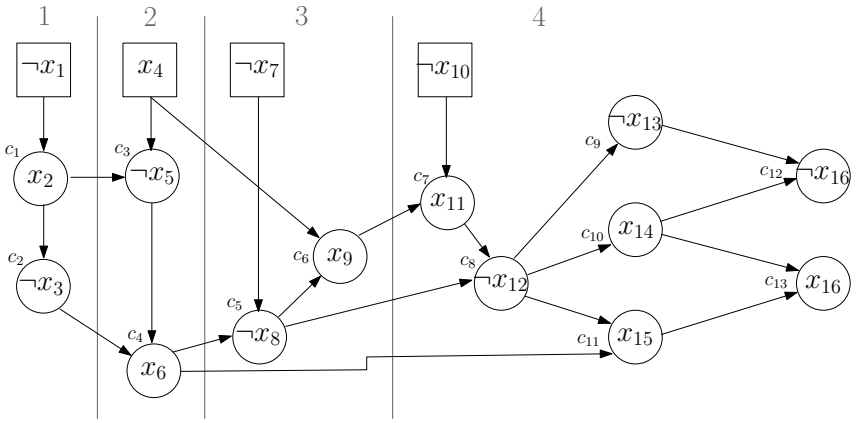


FIGURE 1.2 : *Graphe d'implication*. Les niveaux de décision sont affichés en haut. Les nœuds carrés représentent les littéraux de décision. Les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'entre eux, la clause responsable de cette affectation est annotée. Le conflit se trouve sur la variable x_{16}.

On considère également l'interprétation partielle obtenue par les littéraux de décision $\neg x_1@1$, $x_4@2$, $\neg x_7@3$ et $x_{10}@4$. On a donc $\mathcal{I} = \{\langle \neg x_1@1, x_2@1, \neg x_3@1\rangle, \langle x_4@2, \neg x_5@2, x_6@2\rangle, \langle \neg x_7@3, \neg x_8@3, \neg x_9@3\rangle, \langle \neg x_{10}@4, x_{11}@4, \neg x_{12}@4, \neg x_{13}@4, x_{14}@4, x_{15}@4, x_{16}@4\rangle\}$, Les littéraux grisés dans la formule sont les littéraux falsifiés par l'interprétation $\mathcal{I}$. L'interprétation $\mathcal{I}$ falsifie la formule (clause c_{13}).

Un graphe d'implication est un graphe acyclique dirigé (DAG) associé à l'interprétation courante. Les nœuds d'un tel graphe sont les littéraux affectés à *vrai* par l'interprétation. Il existe une arête entre un nœud x et un nœud y si l'affectation de x à *vrai* implique (à l'aide d'une clause unisatisfaite) l'affectation de y à *vrai*. Les variables de décision n'ont donc pas d'arcs ascendants. La construction d'un tel graphe nécessite de connaître les raisons (clauses) d'affectations des littéraux unitaires tout au long de cette interprétation. La figure 1.2 montre le graphe d'implication pour l'interprétation $\mathcal{I}$.

Ce graphe est exploité avec la notion de UIP (Unique Implication Point). Un UIP est un nœud du dernier niveau de décision (ici le niveau 4) qui domine[5] les deux nœud correspondant à la variable conflictuelle (ici les nœuds x_{16} et $\neg x_{16}$). Il représente une décision alternative provoquant le même conflit. Chaque conflit possède au moins un UIP, la variable de décision du dernier niveau. Dans notre exemple, il y a 3 UIPs : $\neg x_{12}$, x_{11} et $\neg x_{10}$ (le littéral de décision). L'UIP le plus proche du conflit est nommé le premier UIP (ou encore FUIP pour First UIP en anglais), c'est celui-là qui est utilisé dans les démonstrateurs SAT [ZMMM01]. Le nogood responsable du conflit va être généré en effectuant des résolvantes entre les clauses responsables du conflit (utilisé durant la propagation unitaire) en remontant de celui-ci vers la variable de décision du dernier niveau jusqu'à obtenir une clause contenant un seul littéral du dernier niveau de décision (c'est le rôle de la fonction `analyseConflit`). Sur notre exemple, on a :

$$
\begin{aligned}
c_1^{\otimes} &= c_{13} \otimes_{x_{16}} c_{12} &&= x_{13}@4 \vee \neg x_{14}@4 \vee x_{15}@4\\
c_2^{\otimes} &= c_1^{\otimes} \otimes_{x_{15}} c_{11} &&= \neg x_6@2 \vee x_{12}@4 \vee x_{13}@4 \vee \neg x_{14}@4\\
c_3^{\otimes} &= c_2^{\otimes} \otimes_{x_{14}} c_{10} &&= \neg x_6@2 \vee x_{12}@4 \vee x_{13}@4\\
c_4^{\otimes} &= c_3^{\otimes} \otimes_{x_{13}} c_9 &&= \neg x_6@2 \vee x_{12}@4*
\end{aligned}
$$

5. Un nœud a du niveau de décision d domine un nœud b si tout chemin partant de la variable de décision du niveau d vers b doit passer par a [ZMMM01].

La dernière clause générée est la première résolvante possédant un et un seul littéral du dernier niveau de décision (x_{12}) qui est également le premier UIP. C'est cette clause, dite *assertive*, qui va être apprise par le solveur. Non seulement cet apprentissage permettra d'éviter d'atteindre à nouveau le même échec, mais il va également permettre de faire un retour arrière. En effet, la connaissance de cette clause falsifiée au niveau de décision 4, nous permet d'affirmer que x_{12} aurait du être affecté à *vrai* par propagation unitaire dès l'affectation de x_6, donc au niveau 2 (c'est le rôle de la fonction `calculRetourArriere`). Le solveur va donc défaire les affectations des niveaux de décisions 4 et 3 et affecter x_{12} à vrai au niveau 2 en lui donnant comme raison la nouvelle clause $c_4^{\otimes}$. Ainsi, et contrairement à un solveur DPLL classique, le solveur CDCL peut effectuer des sauts arrière (c'est le rôle de la fonction `retourArriere`).

Un travail supplémentaire peut être fait durant cette analyse de conflit. Il est tout d'abord possible de continuer à effectuer des résolutions si celles-ci n'augmentent pas la taille de la clause assertive [SB09]. Plus récemment, deux travaux différents ont montré comment découvrir des clauses sous-sommées durant l'analyse de conflits [HS09, HJS09a]. De notre côté, nous proposons dans la section 3 un cadre étendu à l'analyse de conflits prenant en compte les clauses satisfaites par la formule.

À chaque conflit, les solveurs CDCL apprennent donc une nouvelle Clause dont la taille peut être relativement importante. Dès lors, ils se heurtent à deux problèmes majeurs : l'utilisation de la mémoire et le temps utilisé pour la propagation unitaire. Pour lutter contre cela, la base de clauses apprises est régulièrement nettoyée [ES04, GN02]. Un score, similaire au score de l'heuristique dynamique utilisée pour le choix des variables (voir section suivante), est utilisé pour garder les clauses qui semblent importantes. Néanmoins, ce choix est heuristique et peut amener à supprimer des clauses essentielles pour la suite, ce qui peut s'avérer dramatique. Les solveurs n'ont donc d'autres choix que de laisser le nombre de clauses apprises croître exponentiellement. Nous proposons dans la section 2 une nouvelle mesure statique permettant d'évaluer la qualité des clauses apprises et autorisant une suppression beaucoup plus agressive des mauvais *nogoods*.

1.3 Heuristiques dynamiques

Les structures de données paresseuses introduites dans CHAFF ont provoqué un bouleversement du monde SAT qui dépasse de loin leurs motivations initiales. En effet, ces structures de données ont interdit toute information de type *lookahead* (comme le simple comptage d'occurrences des littéraux). Du coup, toutes les heuristiques utilisées jusqu'alors [JW90, Fre95, DD01] sont devenues obsolètes. Les auteurs de CHAFF [MMZ+01] ont donc proposé une nouvelle heuristique, nommée VSIDS (« *Variable State Independent Decaying Sum*») qui s'est révélée étonnement efficace [6]. Elle ne met pas à jour les informations à chaque affectation, mais à chaque conflit. Chaque variable possède un poids qui augmente à chaque fois que celle-ci est utilisée dans la construction de la clause assertive. À chaque décision, on choisit la variable non affectée possédant le poids le plus important (fonction `choixLittéralDecision`). Ainsi, cette heuristique est très facile à calculer et concentre l'effort de recherche dans la partie la plus récemment conflictuelle. Afin de permettre de diversifier un peu l'espace de recherche, on ajoute un phénomène d'oubli. A chaque conflit, le poids ajouté à chaque variable est augmenté exponentiellement afin de privilégier les variables apparaissant dans les derniers conflits. Cette heuristique, associée aux redémarrages (voir la section suivante), permet de faire remonter les variables les plus conflictuelles en haut de l'arbre de recherche. Des variantes et autres améliorations ont été proposées [GN07], mais le principe de base reste le même.

Une fois la variable de décision choisie, il faut déterminer la polarité de cette dernière (choix de la valeur de la variable). Les auteurs de RSAT [PD07a] ont proposé un mécanisme de sauvegarde de cette

6. On notera qu'une heuristique basée sur ce principe a été proposée avant VSIDS [BGS99]

valeur de vérité lors de l'interprétation précédente. C'est cette valeur qui sera choisie pour affecter les variables de décision. De cette manière, les affectations (a priori) correctes sont conservées malgré le saut arrière. L'amélioration des solveurs par cette technique est significative, surtout dans le cas d'instances satisfaisables.

Enfin, l'initialisation de l'heuristique est également importante. Le plus simple, et pas le moins efficace, consiste, sur la première branche, à choisir les variables dans l'ordre croissant. En effet, lors du codage d'un problème donné vers le problème SAT, les premières variables sont souvent des variables du problème initial, les variables suivantes sont introduites pour éviter une explosion combinatoire lors de la traduction du problème initial vers le problème SAT.

1.4 Redémarrages

A l'origine les redémarrages ont été introduits pour éviter les phénomènes de longue queue [GSK98]. En effet, si un solveur a branché sur les mauvaises variables en haut de l'arbre de recherche, il a très peu de chances de trouver une solution et les redémarrages peuvent lui permettre de corriger ce problème. Ceci est d'autant plus vrai avec les heuristiques dynamiques qui modifient à chaque conflit le poids d'une variable. Aujourd'hui le constat n'est plus exactement le même, les solveurs utilisent des stratégies de redémarrage de plus en plus rapides. L'utilité des redémarrages n'est plus reconnue comme étant d'essayer de chercher ailleurs dans l'espace de recherche, mais plutôt d'atteindre le même espace de recherche avec un chemin et des graphes d'implications différents [Bie08a]. Voici quelques stratégies de redémarrages utilisées par différents solveurs (fonction `redémarrage`) :

- **constante** : SIEGE [Rya04] redémarre tous les 16000 conflits, CHAFF [MMZ+01] (version 2004) tous les 700.
- **géométrique** : MINISAT [ES04] redémarre la première fois à 100 conflits, et utilise une croissance de 50%.
- **luby** : Cette stratégie [LSZ93] suit une loi de croissance moins commune : $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8 \ldots$ avec un coefficient multiplicatif (32 ou 512 dans la plupart des cas). Cette stratégie est optimale si les redémarrages sont indépendants entre eux et si le temps de recherche restant est indéterminé. C'est l'alternance entre redémarrages très rapides et plus lents qui rend cette stratégie très efficace. Elle est utilisée par RSAT [PD07b] ou encore TINISAT [Hua07b].
- **inner-outer** : Cette stratégie est adoptée par PRECOSAT [Bie08a]. Elle ressemble à la stratégie de luby (alternance de redémarrages rapides et lents). Une série géométrique extérieure fixe la borne maximale que peut atteindre la série géométrique intérieure. Les redémarrages sont fixés par la valeur de la série géométrique intérieure.

Toutes ces différentes stratégies de redémarrages ont été comparées dans [Hua07b]. Elles ont toutes en commun d'être pré-determinées. Depuis quelques temps, des tentatives de stratégies de redémarrages dynamiques sont proposées. Elles exploitent différentes informations comme la taille des résolvantes [PD09b], la profondeur moyenne de l'arbre de recherche et des sauts arrière [HJS09b], ou encore le nombre récent de changements de valeur de variables [Bie08b]. Nous introduisons dans la section 2.3 une nouvelle stratégie de redémarrage dynamique.

2 Qualité des clauses apprises

Nous proposons dans cette section une nouvelle mesure permettant d'évaluer la pertinence d'une clause apprise. Cette mesure vient de nombreuses expérimentations menées avec Laurent SIMON et nous commençons cette section par en décrire une synthèse.

2.1 Étude expérimentale

Quinze ans en arrière, la recherche autour des solveurs SAT était guidée par des notions de type *lookahead* [LA97]. Le but était, par exemple, de réduire le plus possible la taille de l'arbre de recherche ; le compromis entre le temps nécessaire à un algorithme de filtrage et l'information filtrée permettait d'évaluer facilement son efficacité. De par leur architecture, les solveurs CDCL sont beaucoup plus difficiles à appréhender. En effet, les structures de données paresseuses empêchent tout comptage de type *lookahead*. Les clauses (raisons) utilisées pour la propagation unitaire influent sur les nogoods générés, et par conséquent, sur l'heuristique de choix des variables. À cause des redémarrages rapides, les sauts arrière influent moins sur la longueur de la preuve qu'auparavant. Ainsi, il est devenu très fréquent de voir des *idées prometteuses* finir dans des cartons à cause d'effets de bords difficilement explicables. Aujourd'hui, nous savons donc implanter des solveurs (très) efficaces, qui plus est en moins de 1000 lignes de code, mais nous ne savons pas toujours quels sont les mécanismes qui sont vraiment importants et pourquoi ils le sont. De nombreuses questions restent en sursis. Que sont de bonnes clauses apprises ? Pourquoi les redémarrages à la luby sont-ils si efficaces ? Est ce que certaines clauses apprises ne servent qu'à mettre à jour les heuristiques et simuler le retour arrière ?

Dans ce contexte, il nous semble de plus en plus important de construire une réelle science expérimentale autour des algorithmes [Hoo94] afin de comprendre, et par conséquent d'améliorer, le comportement des solveurs de type *lookback*. Cela a été le fil conducteur de recherches effectuées avec Laurent SIMON **[AS08, AS09b]** et Mouny SAMY MODELIAR **[ASMS09]** [7]. Nous présentons dans cette section une synthèse de ces travaux.

Nous avons tout d'abord voulu connaître le réel impact de la taille des clauses apprises sur les solveurs CDCL. Pour cela, nous avons réalisé l'expérimentation suivante : nous avons choisi 150 instances provenant de compétitions antérieures. Nous avons ensuite lancé MINISAT [ES04] sur ces instances en oubliant dès que possible (lorsqu'elles ne sont plus utilisées comme raison) 25% des clauses apprises de n'importe quelle taille. Dans un second (resp. troisième) temps, nous avons relancé MINISAT, mais en oubliant cette fois-ci 50% des petites (resp. grandes) clauses apprises. Le but est d'oublier le même nombre de clauses apprises, mais avec des caractéristiques différentes. La figure 1.3 montre, à l'aide d'une figure *cactus*, les résultats obtenus. Contrairement aux idées reçues, il semble que les petites clauses ne sont pas significativement plus importantes que les grandes clauses. En effet, les 3 courbes sont relativement identiques. Nous avons réalisé les mêmes expérimentations avec d'autres paramètres, comme le nombre de résolutions effectuées pour générer la clause, la profondeur de la résolution... et les résultats ont été similaires. Ces premières expérimentations ont donc montré la difficulté à déterminer, à l'aide d'une mesure statique, quelles sont les clauses apprises importantes.

Nous avons également mené d'autres types d'expérimentations visant à observer l'évolution de certains paramètres durant la recherche. Nous avons par exemple mis en évidence que, sur une grande majorité d'instances, les niveaux de décisions décroissent tout au long de la recherche et que cette décroissance semble reliée avec l'efficacité (ou inefficacité) des solveurs CDCL **[AS09b]**. Nous avons mené les expérimentations suivantes : pour chaque instance, nous enregistrons à quel niveau de décision (variable d de l'algorithme refalg :cdcl) se produit chaque conflit (variable nb_c de l'algorithme 1.1. Nous calculons ensuite une simple régression linéaire sur l'ensemble de points (nb_c, d). Cette régression nous donne l'évolution (croissance ou décroissance) des niveaux de décision durant la recherche. Tout cela est résumé dans le tableau de la figure 1.4. Dans 83% des instances testées, on observe une décroissance, qui plus est, sans que cela soit lié à leur satisfaisabilité. D'un premier abord, ceci peut sembler logique. En effet, entre le rôle de l'heuristique VSIDS et le fait que des contraintes sont ajoutés tout au long de la

7. Afin de différencier rapidement l'état de l'art des publications où j'interviens, ces dernières seront en caractères gras.

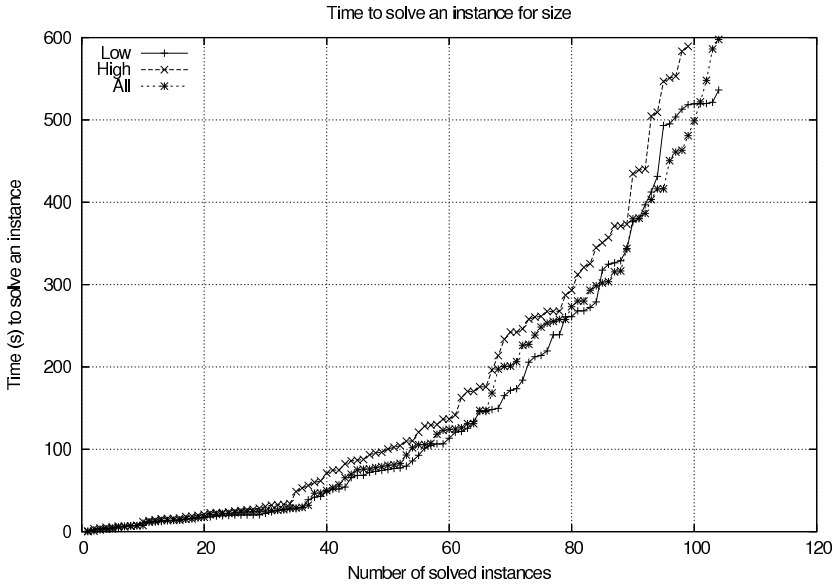


FIGURE 1.3 : *Impact sur les performances de* MINISAT *de la suppression des clauses selon leur taille*. L'axe des x correspond au nombre d'instances résolus, l'axe des y au temps qu'il faut pour les résoudre. Dans la première courbe on supprime au hasard 25% de toutes les clauses apprises (*All*). Dans la seconde, 50 % des petites clauses (*Low*). Dans la dernière, 50% des grandes (*High*). L'axe des x représente le nombre d'instances résolues et l'axe des y le temps nécessaire pour les résoudre.

recherche, on pourrait penser que l'on va propager de plus en plus de littéraux unitaires et arriver de plus en plus rapidement à un conflit. On voit tout de même que cela n'est pas le cas pour toutes les instances, et que certaines familles ont un comportement totalement différent. Ces diverses expérimentations nous ont mené à proposer une mesure statique permettant d'évaluer l'importance d'une clause apprise.

2.2 Une mesure statique

Nous avons également mis en évidence une relation entre la vitesse de décroissance des niveaux de décision tout au long de la recherche et l'efficacité de solveurs. On peut donc supposer que l'amélioration de la décroissance permettra d'améliorer l'efficacité des solveurs. Notons que cela avait déjà été pointé du doigt en 2001 par les auteurs de CHAFF : "*A good learning scheme should reduce the number of decisions needed to solve certain problems as much as possible*" [ZMMM01], mais nous en donnons ici une explication plausible.

Rappelons que durant la recherche chaque littéral de décision est suivi de blocs de propagations unitaires. Une *bonne* clause apprise devrait lier deux blocs entre eux afin qu'ils ne deviennent qu'un. À partir de toutes ces expérimentations et considérations nous avons proposé une mesure statique permettant d'évaluer la qualité d'une clause apprise.

Définition 1.16 (Distance LBD (« *Literal Block Distance* »)) Soient une formule CNF Σ, une interprétation partielle $\mathcal{I}_p$ conflictuelle et c la clause assertive construite à partir de $\mathcal{I}_p$. Alors le LBD de c est égal au nombre de niveaux de décision différents des littéraux de la clause c.

Si on reprend l'exemple de la section 1.2, la clause assertive générée est $c_4^{\otimes} = (\neg x_6@2 \vee x_{12}@4)$, son LBD est donc égal à 2. Cette mesure se calcule lors de la création de la clause assertive. Intuitivement, il

FAMILLE	#INSTANCES	%DECR
een	8	62%
gold	11	100%
grieu	7	71%
hoons	5	100%
ibm-2002	7	71%
ibm-2004	13	92%
manol-pipe	55	91%
mizh	13	0%
schup	5	80%
simon	10	90%
vange	3	66%
velev	54	92%
TOTAL	199	83%

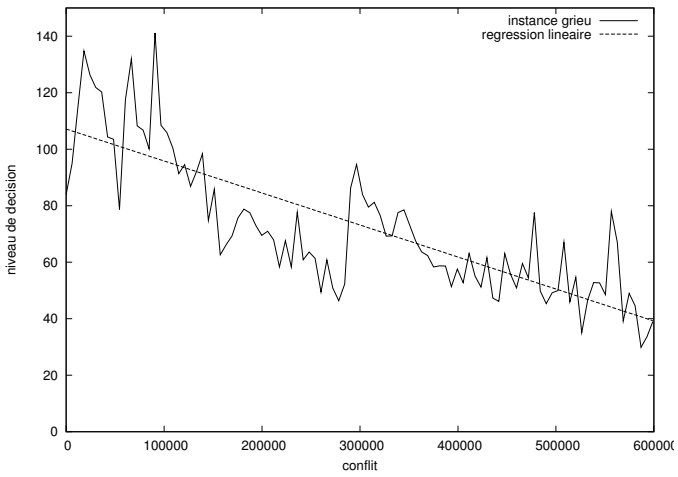


FIGURE 1.4 : *Décroissance des niveaux de décision*. Le tableau de gauche reporte, pour chaque famille d'instance, le nombre d'instances dans la famille et le pourcentage d'entre elles exhibant une décroissance des niveaux de décision. La figure de droite montre un exemple (sur une instance de la famille grieu) de cette décroissance des niveaux de décision au cours de la recherche avec la régression linéaire associée .

est facile de comprendre l'utilité des clauses dont le LBD est égal à 2.

Les littéraux qui la composent sont dans deux niveau de décision et donc dans deux blocs de propagations différents : le dernier puisqu'elle est assertive, et un autre, celui vers lequel va être effectué le retour arrière. Lors des prochaines propagations des mêmes littéraux, elles vont pouvoir coller ces deux blocs de propagation ensemble de tele sorte qu'il n'en fassent plus qu'un, quelle que soit la taille de la clause.

Ce sont ces clauses que nous supposons très importantes. Nous leur avons donné un nom spécial : les *clauses glues*. De plus, nous montrons dans **[AS09b]**, que plus une clause a un grand LBD, moins elle est utilisée dans le reste de la preuve (comme clause raison dans la propagation). Enfin, Saïd JABBOUR a également montré dans sa thèse [Jab08] que, parmi tous les schémas d'apprentissage, la mesure LBD est optimale pour le FUIP. Cela peut être une nouvelle explication de l'efficacité du FUIP parmi les autres schémas d'apprentissage [ZMMM01].

2.3 Intégration au sein d'un solveur SAT

Pour montrer la pertinence de notre mesure, nous l'avons intégrée au sein d'un solveur CDCL en prenant comme base le solveur MINISAT. Nous avons nommé le solveur résultant GLUCOSE **[AS09b, AS09a]**, pour sa propension à détecter et conserver des clauses glues. Voici quelques unes de ses fonctionnalités.

- Considérant que la mesure LBD est importante et précise, nous avons opté pour une suppression agressive des clauses apprises. Quelle que soit la taille de la formule, et en conservant de préférence les clauses de petit LBD, nous supprimons la moitié des clauses apprises tous les $20000 + 500 \times x$ conflits (x étant le nombre de fois où la réduction des clauses a déjà été réalisée). Les clauses avec un grand LBD sont donc supprimées assez rapidement. Les performances de la propagation unitaire sont grandement améliorées.
- Nous avons mis au point une stratégie de redémarrage dynamique qui essaie de favoriser la dé-

solveur		total	instance SAT	instances UNSAT	temps
PRECOSAT	[Bie08a]	**204**	**79**	125	**180 345 s**
GLUCOSE		**204**	77	**127**	218 826 s
LySAT	[HJS09c]	197	73	124	198 491 s
RSAT (premier en 2007)	[PD07b]	180	69	111	195 748 s
VBS		229	91	138	153 127 s

TABLE 1.1 : Comparaison de PRECOSAT, GLUCOSE, RSAT sur les 292 instances de la catégorie application issue de la compétition SAT'09. Le temps limite est fixé à 10 000 secondes. Pour chaque solveur, on reporte le nombre d'instances SAT, UNSAT et totales résolues et le temps en secondes cumulé nécessaire pour les résoudre. Le solveur VBS simule le meilleur solveur théorique.

croissance des niveaux de décision durant la recherche : si ce n'est pas le cas, alors un redémarrage est effectué. Pour cela, on calcule la moyenne (glissante) des niveaux de décision sur les 100 derniers conflits, si cette dernière est plus grande que 0.7 fois la moyenne de tous les niveaux de décision depuis le début de la recherche alors on procède à un redémarrage.

– L'heuristique VSIDS est modifiée afin de récompenser certaines variables propices à générer des clauses de petit LBD.

Notre solveur GLUCOSE a participé à la compétition organisée en marge de la conférence SAT'09 (voir `http://www.satcompetition.org`). Il y a obtenu deux distinctions : premier dans la catégorie application sur les instances insatisfaisables et deuxième dans la catégorie application sur toutes les instances. Le tableau 1.1 montre une synthèse des résultats obtenus par les trois premiers lors de cette compétition. Afin de mettre en évidence les progrès réalisés en deux ans, nous avons ajouté RSAT, le vainqueur en 2007. Nous avons également mis ce que l'on nomme le VBS (*Virtual Best Solver*), c'est le solveur théorique correspondant aux meilleures réponses données par l'ensemble des solveurs. On note donc que sur les 292 instances de la compétition, seuls 229 ont été résolues. Les progrès à accomplir sont donc encore énormes.

3 Un cadre étendu pour l'analyse des conflits

Parmi les composantes essentielles des solveurs CDCL, l'analyse des conflits et, par conséquent, le graphe d'implication tiennent une place importante (voir section 1.2). Néanmoins, ce graphe d'implication est construit de manière incomplète et ne propose qu'une vue partielle des implications entre les littéraux. En effet, par souci d'efficacité, seule la première clause (raison) trouvée permettant de propager un littéral unitaire est enregistrée. De plus, les clauses déjà satisfaites sont ignorées durant le processus de propagation unitaire.

Durant la thèse de Saïd JABBOUR, co-encadrée par Lakhdar SAÏS et moi-même, nous avons collaboré avec Lucas BORDEAUX et Youssef HAMADI et avons proposé ensemble une extension des graphes d'implications contenant des arcs additionnels appelés arcs arrières et obtenus en tenant compte des clauses satisfaites **[ABH+08a, ABH+08b]**. Nous présentons ci-dessous et à l'aide d'un exemple cette notion. Pour une définition plus formelle, le lecteur pourra se référer à la thèse de Saïd JABBOUR [Jab08].

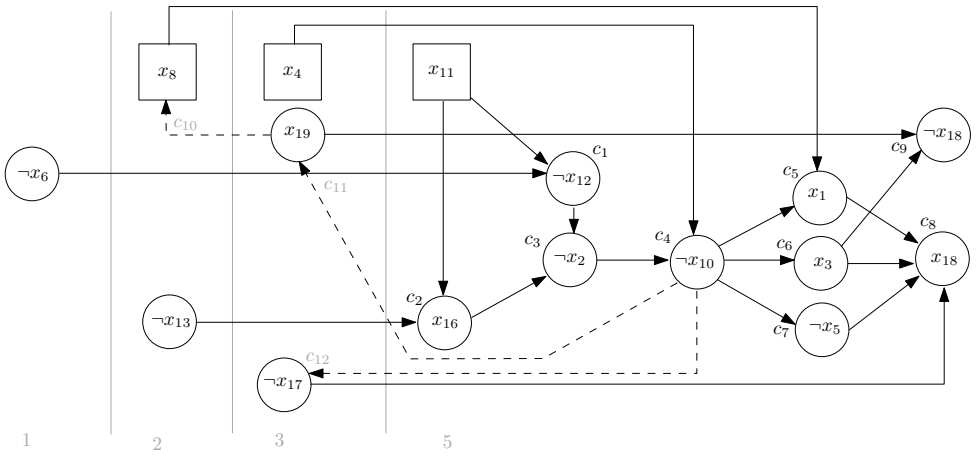


FIGURE 1.5 : *Graphe d'implication étendu.* Les nœuds carrés représentent les points de choix, les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'eux la clause responsable de cette affectation est annotée. Les arcs en pointillé représentent les arcs inverses, les clauses associées sont de couleur grise.

On considère la formule CNF contenant, entre autre, les clauses 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_9)	$\neg x_3 \vee \neg x_{19} \vee \neg x_{18}$
(c_{10})	$x_8 \vee \neg x_{19}$	(c_{11})	$x_{19} \vee x_{10}$	(c_{12})	$\neg x_{17} \vee x_{10}$

On considère également l'interprétation partielle contenant, entre autre les littéraux de décision $x_8@2$, $x_4@3$ et $x_{11}@5$, $\mathcal{I} = \{\langle \ldots \neg x_6@1 \ldots \neg x_{17}@1 \rangle \langle x_8@2 \ldots \neg x_{13}@2 \ldots \rangle \langle x_4@3 \ldots x_{19}@3 \ldots \rangle \ldots \langle x_{11}@5 \ldots \rangle\}$. Le niveau de décision courant est 5 et l'interprétation $\mathcal{I}$ falsifie la formule. L'observation des clauses de la formule montre que l'ensemble des clauses $c_1 \ldots c_8$ a été utilisé lors de la propagation unitaire. La clause c_9 est falsifiée et les clauses c_{10}, c_{11} et c_{12} sont satisfaites par $\mathcal{I}$ respectivement par $x_8@2$, $x_{19}@3$ et $\neg x_{17}@1$. De plus, ces trois dernières clauses sont particulières, elles sont unisatisfaites et les littéraux falsifiés le sont à un niveau supérieur du littéral qui les satisfait (autrement elles auraient été utilisées lors de la propagation unitaire). Ce sont ces clauses, appelées *arcs inverses* et ignorées habituellement, qui sont rajoutées au graphe d'implication et qui servent durant le processus d'analyse de conflit. La figure 1.5 montre le graphe d'implications augmenté des arcs inverses (en pointillé). Notez quelque chose d'inhabituel dans ce graphe : les points de choix peuvent également avoir des arcs entrants (voir x_8), ce qui est impossible avec un graphe d'implication classique.

Lorsque l'on effectue l'analyse de conflit classique sur ce graphe on obtient la clause assertive $c^{\otimes} = (x_{17} \vee \neg x_{19} \vee \neg x_8 \vee x_{10})$. Les arcs inverses peuvent être utilisés et permettent de continuer le processus de résolution :

$$
\begin{aligned}
c_1^{\otimes} &= c^{\otimes} \otimes_{x_8} c_{10} &&= x_{17} \vee \neg x_{19} \vee x_{10} \\
c_2^{\otimes} &= c_1^{\otimes} \otimes_{x_{19}} c_{11} &&= x_{17} \vee x_{10} \\
c_3^{\otimes} &= c_2^{\otimes} \otimes_{x_{17}} c_{12} &&= x_{10}
\end{aligned}
$$

On obtient donc une nouvelle clause assertive qui, dans le cas présent, sous-somme la clause assertive $c^{\otimes}$ et permet un saut arrière plus important. Le processus est identique si on désire utiliser un autre schéma d'apprentissage que le FUIP.

Nécessitant le parcours de la partie satisfaite de la formule, la recherche d'arcs inverses est une opération qui peut s'avérer coûteuse. De plus, la génération de nouvelles résolvantes peut dégrader la clause assertive initiale. Pour éviter ceci, nous avons proposé une première intégration des arcs inverses qui permet d'effectuer un saut arrière plus important. Nous nous limitons aux clauses assertives du type : $c = l \vee l_1@b \vee l_2@k_2 \dots l_n@k_n$ avec l le littéral assertif, b le niveau de retour arrière et tous les autres littéraux avec un niveau $k_i < b$. Les arcs arrières vont être utilisés pour opérer une résolution sur le littéral l_1. L'arc inverse aura également certaines caractéristiques précises. Ses littéraux du dernier niveau de décision ont été visités durant l'analyse de conflit et les autres ont un niveau de décision strictement inférieur à b. Si de tels arcs arrières existent, la résolution entre la clause assertive initiale et ces derniers va générer une nouvelle clause assertive ne contenant pas le littéral $l_1@b$ et introduisant uniquement des littéraux de niveaux strictement inférieurs à b. Ainsi, le saut arrière induit par la nouvelle clause assertive est plus important que celui initialement prévu. Les résultats obtenus avec cette première intégration donnent de très bons résultats sur certaines familles d'instances. Toutefois, avec l'utilisation des redémarrages dans les solveurs SAT, il faut noter que l'amélioration des sauts arrière a moins d'importance qu'auparavant. La notion de graphe de conflit étendu n'en reste pas moins importante. Nous pensons par exemple l'utiliser pour obtenir des clauses assertives avec des meilleurs LBD.

4 Résolution étendue

4.1 Systèmes de preuves

Un système de preuve (propositionel) [CR79] est un algorithme en temps polynomial SP tel que pour toute formule propositionelle insatisfaisable Σ, il existe une preuve p telle que SP accepte l'entrée (Σ, p), c'est à dire que SP vérifie que p est bien une preuve d'insatisfaisabilité pour Σ. En d'autres termes, un système de preuve est un moyen efficace de vérifier la validité d'une preuve d'insatisfaisabilité. Mais trouver une telle preuve reste difficile dans le cas général. Les systèmes de preuves ont été étudiés en informatique théorique comme une façon de comparer les problèmes NP et co-NP. Ils permettent également de comparer les systèmes d'inférences les uns par rapport aux autres en permettant par exemple d'établir une hiérarchie de ces derniers reflétant leurs puissances relatives.

Un système de preuves SP admet des preuves courtes (resp. longues) pour une famille d'instances $\mathcal{F}$ si la taille de la plus petite preuve croît polynomialement (resp. exponentiellement) avec la taille des instances de $\mathcal{F}$. On dit alors que $\mathcal{F}$ est simple (resp. difficile) pour SP.

Un système de preuves SP_1 p-simule un système de preuves SP_2 si pour toute formule insatisfaisable Σ la plus petite preuve de Σ dans SP_1 est au pire polynomialement plus grande que la plus petite preuve de Σ dans SP_2. S'ils se p-simulent l'un et l'autre, ils sont alors polynomialement équivalents. La notion de p-simulation exprime une hierarchie entre les systèmes de preuves. En effet, si SP_1 p-simule SP_2 alors SP_1 est au moins aussi puissant que SP_2.

Le système de preuve le plus connu et le plus utilisé est le système de preuve par résolution [Rob65] (RES). Une preuve dans ce système est une suite de clauses $\langle c_1 \dots c_m, c_{m+1}, \dots, c_s = \bot \rangle$ où les m premières sont celles de la formule initiale et les clauses suivantes sont produites par résolution, i.e., la clause $c_k = c_i \otimes_l c_j$ avec $i < j < k$, la clause c_s est la clause vide.

La trace d'exécution d'un solveur CDCL peut se voir comme un système de preuves [BKS04, VG05,

HBPG08, PD09a] qui, dès lors, permet de p-simuler RES. Ce résultat est d'autant plus important que les autres algorithmes de résolution pour SAT, y compris la version originelle de DPLL [DLL62] sont connus pour être plus faibles que la résolution. Par exemple, la procédure DPLL peut-être p-simulée par une restriction de la résolution où chaque clause qui n'appartient pas à la formule de départ ne peut être utilisée qu'une fois durant la preuve. Les bonnes performances pratiques des solveurs SAT modernes peuvent être expliquées, au moins en partie, par ce résultat théorique.

Néanmoins, le système de preuves RES a ses propres limites. Plusieurs familles d'instances ont été prouvé comme étant difficile pour ce système de preuve, comme le problème des pigeons [Hak85] ou encore [Urq87, CS88].

Un autre système de preuve bien connu est la résolution étendue ER [Tse70]. C'est une extension de la résolution à laquelle on ajoute la règle suivante : à chaque étape il est possible d'ajouter un lemme dans la formule, c'est-à-dire d'ajouter une nouvelle variable z (qui n'apparaît pas dans la formule) ainsi que les clauses qui codent $z \Leftrightarrow l_1 \vee l_2$ où l_1 et l_2 sont deux littéraux de la formule (ce lemme est équivalent aux trois clauses $(\neg w \vee l_1 \vee l_2)$, $(z \vee \neg l_1)$, $(z \vee \neg l_2)$). Une preuve dans ER est donc une suite $\langle c_1 \dots c_m, \{c|e\}_{m+1}, \dots, c_s = \bot \rangle$ où, pour $m < i \leq s$, c_i est une clause obtenue par résolution et e_i est l'ajout d'une nouvelle variable par la règle mentionnée ci-dessus. Sans perte de généralité, il est possible d'introduire l'ensemble des nouvelles variables au début de la preuve, qui peut donc être réécrite ainsi : $\langle c_1 \dots c_m, e_{m+1}, \dots e_{m+k}, c_{m+k+1}, \dots, c_s = \bot \rangle$. Tout en étant très simple, l'ajout de nouvelles variables permet à la résolution étendue d'être un système de preuve très efficace : aucun système de preuve n'a été prouvé comme étant plus fort que la résolution étendue. Ainsi, dans un *magnifique* article [Coo76], Stephen COOK montre comment utiliser la résolution étendue pour construire une preuve polynomiale des pigeons prenant en compte le coté inductif de ce problème.

Cette puissance ne va pas sans contre-partie. Il est en effet très difficile de trouver judicieusement et automatiquement les bonnes paires de littéraux à étendre. C'est certainement ce qui explique l'absence de la résolution étendue dans les solveurs SAT actuels. À titre d'exemple, dans [Coo76], Stephen COOK utilise la sémantique du problème pour choisir les lemmes à ajouter. Avec Laurent SIMON et George KATSIRELOS, nous avons proposé une première intégration de la résolution étendue **[AKS10]** que nous développons dans la section suivante.

4.2 Une forme restrictive de la résolution étendue

Nous avons proposé une simple restriction à ER limitant de manière significative le nombre potentiel de lemmes à ajouter à la formule et pouvant expliquer pourquoi ER peut aider à créer des preuves courtes. C'est cette restriction que nous présentons ci-dessous.

Soient deux littéraux l_1 et l_2 provenant de deux variables différentes. Supposons que nous ayons appris durant la recherche les clauses $c_i = \neg l_1 \vee \alpha$ et $c_j = \neg l_2 \vee \alpha$ où α est une disjonction de littéraux. Supposons enfin que ces clauses sont toutes les deux utilisées dans un processus de résolution avec la même séquence de clauses $c_{m_1}, \dots, c_{m_k}$ afin de dériver $c'_i = \neg l_1 \vee \beta$ et $c'_j = \neg l_2 \vee \beta$. Dans ce cas, nous pouvons tirer bénéfice d'étendre la formule initiale avec la nouvelle variable $x \iff l_1 \vee l_2$. En effet, nous pouvons alors effectuer une résolution entre c_i et c_j avec $\neg x \vee l_1 \vee l_2$ afin d'obtenir $c_k = \neg x \vee \alpha$. La clause c_k peut alors être résolue avec $c_{m_1}, \dots, c_{m_k}$ pour dériver $\neg x \vee \beta$. Ainsi toute étape dans la résolution qui soit commune avec c'_i et c'_j ne sera pas dupliquée permettant une *compression* de la preuve.

Notons que dans ce schéma, les clauses c_i et c_j n'ont pas besoin d'être des clauses de la formule initiale, mais peuvent être des clauses dérivées. C'est important, car cela nous permettra d'introduire des variables en examinant la preuve au fur et à mesure de sa construction. La définition suivante formalise une généralisation de ce schéma comme un système de preuve.

Définition 1.17 (Résolution étendue locale) LER est le système de preuve propositionel qui inclut la règle de résolution mais aussi la règle de résolution étendue telle que à une étape k on peut introduire la variable $z \iff l_1 \vee l_2$ s'il existe deux clauses $c_i \equiv \neg l_1 \vee \alpha$ et $c_j \equiv \neg l_2 \vee \beta$ avec $i < k, j < k$, où α et β sont des disjonctions vérifiant $l \in \alpha \implies \neg l \notin \beta$.

La grande différence entre LER et ER est que la première nécessite des clauses d'une forme donnée afin d'introduire des nouvelles variables. Et de telles clauses peuvent ne pas être dérivées durant la recherche. L'avantage de LER est que les preuves ainsi générées sont moins imprévisibles que celles de ER. En particulier, chaque variable introduite est directement reliée à la preuve.

Néanmoins, la restriction de LER n'est pas trop sévère. En fait, il est facile de vérifier que des problèmes qui sont difficiles pour RES et qui acceptent des preuves polynomiales pour ER peuvent être transformées en des preuves polynomiales pour LER. C'est le cas par exemple pour la preuve ER de Stephen COOK sur le problème des pigeons [Coo76]. Ainsi, LER est plus fort que la résolution.

D'un autre coté, il peut être difficile de générer les deux clauses nécessaires à l'introduction d'une nouvelle variable. La conséquence de cela est que la preuve minimale pour LER peut être plus grande que la preuve minimale pour ER. Actuellement, nous ne savons pas encore s'il existe des familles d'instances difficiles pour LER, mais faciles pour ER ou si LER p-simule ER. C'est l'objet de recherches actuelles.

4.3 Intégration dans un solveur CDCL

Le schéma expliqué dans la section précédente restreint le nombre de paires de littéraux candidates pour la résolution étendue. Malgré tout,ce nombre risque d'être toujours trop élevé pour pouvoir étendre la formule avec tous les candidats. Nous avons donc implanté une version plus restrictive de LER en focalisant notre attention sur les clauses assertives consécutives de la forme $\neg l_1 \vee \alpha$ et $\neg l_2 \vee \alpha$ avec l_1 et l_2 les deux littéraux assertifs associés. De plus, nous avons implanté les stratégies suivantes :

- **Applications successives de la résolution étendue :** Si n clauses apprises successives sont de la forme $c_i = l_i \vee \alpha$ $(1 \leq i \leq n)$ alors nous étendons la formule avec $z_1 \iff \neg l_1 \vee \neg l_2$, et aussi avec $z_i \iff \neg l_i \vee \neg l_{i+1}$ pour $2 \leq i < n$.
- **Injection des variables étendues dans les nouvelles clauses :** A chaque fois qu'une nouvelle variable $z \iff l_1 \vee l_2$ est introduite, nous devons nous assurer que toutes les futures clauses de la forme $l_1 \vee l_2 \vee \beta$ seront remplacées par $z \vee \beta$. Ce remplacement systématique est important. En effet, il permet à la nouvelle variable z d'être propagée même si cela n'était pas le cas pour la clause initiale. Par exemple, si la sous-clause β est fausse, la clause $l_1 \vee l_2 \vee \beta$ n'est pas unitaire, mais la clause $z \vee \beta$ l'est. Les tests expérimentaux ont montré que sans cela, la variable z n'apparaît pas dans l'analyse de conflit et ne participe donc pas à la preuve.
- **Suppression des variables étendues inutiles :** Comme c'est le cas pour les clauses apprises, nous supprimons les variables étendues qui nous semblent inutiles. Ainsi, à chaque nettoyage des clauses apprises, nous supprimons également la moitié des variables étendues, celles qui ont un petit score VSIDS, et qui ont donc peu de chance de participer à la preuve.

L'implantation faite sur les solveurs MINISAT et GLUCOSE a montré des premiers résultats très prometteurs améliorant les performances de leurs solveurs de base sur de nombreuses instances. De plus, des instances habituellement difficiles pour ces solveurs, telles les instances de Urquhart [Urq87] ont pu être résolues assez facilement. De plus amples détails sur les expérimentations peuvent être obtenus sur **[AKS10]**. L'objectif principal, à savoir montrer que l'on peut augmenter la puissance de raisonnement des solveurs CDCL tout en gardant de bonnes performances générales, a été atteint. Toutefois, de nombreuses questions restent ouvertes, nous en reparlerons au niveau des perspectives.

C Solveurs basés sur la recherche locale

Les algorithmes de recherche locale stochastiques sont utilisés depuis les années 70 pour la résolution de problèmes d'optimisation (partitionnement de graphe [LK70] ou encore voyageur du commerce [LK73]). Les premières tentatives pour appliquer ce type d'algorithme au problème SAT ont été réalisées sur un problème particulier, le problème des reines [MJPL90]. Mais c'est en 1992 que la première méthode de recherche locale stochastique dédiée à SAT voit le jour avec GSAT [SLM92]. Depuis, de nombreux travaux connexes ont été réalisés : utiliser la propagation unitaire [HK05], les dépendances fonctionnelles [PTS07], mise en évidence de liens avec des problèmes de physique [MZ02]. . . . Il est très important de noter que les méthodes de recherche locales sont des méthodes incomplètes : elles ne fournissent pas nécessairement de réponse, et dans la plupart des cas, elles ne savent répondre que dans le cas où la formule est satisfaisable. Nos travaux autour de la recherche locale ont surtout eu pour but de proposer des méthodes permettant de prouver l'insatisfaisabilité. Néanmoins, pour plus de clarté, nous commençons par donner le schéma général des méthodes de recherche locale pour SAT. Ensuite, nous introduisons les quelques travaux ayant eu trait à la recherche locale pour l'insatisfaisabilité, avant de présenter nos travaux.

1 Recherche locale pour la satisfaisabilité

Le schéma de recherche locale pour prouver la satisfaisabilité d'une instance SAT se compose de deux phases distinctes : l'intensification et la diversification. La première phase consiste à se déplacer de manière stochastique sur l'ensemble des interprétations complètes de la formule à résoudre. À chaque étape on essaie, par l'inversion de la valeur de vérité d'une variable, de réduire le nombre de clauses falsifiées (phase de descente). Le but est donc de se rapprocher d'un modèle de la formule. La prochaine interprétation est choisie parmi l'ensemble des interprétations voisines (c'est-à-dire différant uniquement sur la valeur d'une seule variable) de l'interprétation courante. Lorsqu'aucune interprétation voisine ne permet de diminuer le nombre de clauses falsifiées, on se trouve alors dans un minimum local et une stratégie d'échappement est utilisée pour en sortir. Afin de ne pas rester confiné dans un sous-espace restreint de l'ensemble des interprétations, la diversification consiste, après un nombre déterminé d'essais infructueux, à générer aléatoirement une nouvelle interprétation complète.

L'algorithme 1.2 présente le schéma général d'un algorithme de recherche locale de type WSAT [SKC94]. Cet algorithme commence par générer aléatoirement une interprétation complète et va ensuite effectuer des modifications locales. Pour ce faire, une clause falsifiée par l'interprétation courante est choisie (ligne 7). S'il existe une variable appartenant à cette clause permettant de diminuer le nombre de clauses fausses (descente), on inverse (« *flip* ») sa valeur de vérité (ligne 9). Sinon, on se trouve dans un minimum local et un critère d'échappement est utilisé (ligne 12). Il est important de noter qu'ici, le minimum n'est pas un «vrai» minimum. En effet, les variables candidates aux flips ne sont qu'un sous-ensemble de l'ensemble des variables falsifiant les clauses. Néanmoins, en ne parcourant pas tous les littéraux possibles mais uniquement une très petite partie d'entre eux, on obtient une méthode beaucoup plus efficace en pratique que la méthode GSAT. Afin de diversifier la recherche, au bout d'un certain nombre de réparations locales ($maxflips$), l'algorithme génère aléatoirement une nouvelle interprétation complète. Ce processus peut être répété un certain de nombre de fois ($maxtries$) fixé au départ.

La stratégie d'échappement utilisée a un grand impact sur l'efficacité des méthodes de recherche locale. Certains de nos travaux sont en relation avec la stratégie d'échappement utilisée. Nous listons donc ci-dessous quelques méthodes proposées dans la littérature :

– **Random Walk :** C'est la stratégie originelle de la méthode WSAT [SKC94]. Elle va flipper la va-

ALGORITHME 1.2 : Recherche locale stochastique

Input : Σ une formule CNF
Output : SAT si Σ est prouvé satisfiable, UNKNOWN si pas de modèle trouvé

```
1  for i=1 to maxtries do
2      I_c ← interpretationComplete(Σ);
3      for j=1 to maxflips do
4          if (I_c est un modèle de Σ) then
5              return SAT                                   // Succès
6          end
7          c ← choisirClauseFalsifiee(Σ, I_c);             // c ∩ I_c = ∅
8          if ∃l ∈ c permettant une descente then
9              I_c = (I_c − {l}) ∪ {¬l};                   // inverser la valeur de l
10         end
11         else
12             l ←sortirMinimumLocal();
13             I_c = I_c − {l} ∪ {¬l};                     // inverser la valeur de l
14         end
15     end
16 end
17 return UNKNOWN;
```

riable qui augmente le moins possible le nombre de clauses falsifiées (mouvement glouton). Néanmoins pour diversifier un peu la recherche, il est possible suivant une probabilité fixée au départ de choisir aléatoirement la variable à flipper dans la clause c (mouvement aléatoire). Une amélioration a ensuite été proposée dans [Hoo02], où les auteurs adaptent dynamiquement la probabilité d'effectuer un mouvement aléatoire. D'autres variantes à cette méthode existent [MSK97, Hoo99].

– **Méthode tabou :** Venant des méthodes de recherche opérationnelle [Glo89, Glo90]. Elle a ensuite été adaptée au problème SAT [MSG95, MSG97]. Le but est d'éviter d'explorer les mêmes interprétations dans un futur proche. Pour cela, les dernières variables flippées sont mises dans une liste tabou (de type FIFO) de taille pré-définie. Une variable de cette liste ne pourra donc pas être flippée. Divers travaux ont été réalisés pour essayer optimiser la longueur de la liste tabou, statiquement [MSG98] ou dynamiquement [BP96].
– **Méthodes dynamiques :** En 1993, Paul MORRIS [Mor93] a proposé une méthode de recherche locale originale. En effet, il propose de modifier dynamiquement la fonction objectif. Il ajoute donc un poids aux clauses. Plutôt que d'essayer de réduire le nombre de clauses non satisfaites, la méthode «*breakout* » essaie de minimiser la somme des poids des clauses. Au début de la recherche, les poids sont tous initialisés à 1. A chaque fois qu'une clause est falsifiée, son poids est augmenté. De cette manière, les minima ont tendance à disparaître. Cette méthode a ensuite été améliorée en ajoutant diverses probabilités [HTH02]. Une autre variante [CI96], beaucoup plus intéressante, consiste à ajouter des résolvantes pour augmenter artificiellement le poids de certaines clauses. Nous y reviendrons dans la section 2.2.

Les méthodes de recherche locale sont une réelle alternative aux méthodes de recherche systématiques. Elles ont montré leur intérêt dans de nombreux domaines (recherche opérationnelle, optimisation...). Néanmoins elles souffrent de plusieurs défauts rédhibitoires. Tout d'abord, dans le cas du problème SAT, les méthodes de recherche locales ont surtout montré leur efficacité pour résoudre des problèmes aléatoires. Une des causes montrées du doigt par les organisateurs eux-mêmes étant la compéti-

ALGORITHME 1.3 : La méthode GUNSAT

Input : Σ une formule CNF
Output : UNSAT si une preuve de $\bot$ est trouvée, UNKNOWN sinon

```
1  for i=1 to MaxTries do
2      for j=1 to MaxFlips do
3          if (2-saturation(Σ)=UNSAT) then return UNSAT ;
4          if (|Σ| > MaxSize) then supprimerClause(Σ);
5          ajouterClause(Σ);
6          resolutionEtendue(Σ);
7          lookAhead(Σ);
8      end
9      reduire(Σ);
10 end
11 return UNKNOWN;
```

tion SAT [LBRS09]. Ainsi, le challenge n°6 proposé Bart SELMAN *et al.* [SKM97] : *« Improve stochastic local search on structural problems by efficiently handling variables dependencies »* est toujours d'actualité aujourd'hui. L'autre défaut, plus structurel celui-là, est que dans la plupart des cas, les méthodes de recherche locale ne peuvent prouver que la satisfaisabilité des instances. Là encore, un challenge a été proposé [SKM97] :« *Design a pratical local search procedure for proving unsatisfiability* ». Quelques tentatives, que nous introduisons dans la section suivante, ont été proposées dans le passé. Un de nos axes de recherche consiste justement à proposer de nouvelles méthodes de recherche locale pour s'attaquer à ces deux challenges très difficiles. C'est l'objet des sections suivantes.

2 Recherche locale pour l'insatisfaisabilité

Les travaux autour de la recherche locale pour l'insatisfaisabilité sont peu nombreux. La première méthode connue est celle proposée par CHA et IWAMA dans laquelle les auteurs ajoutent des résolvantes pour sortir des minima locaux [CI96]. Cette méthode a ensuite été améliorée dans deux travaux distincts [FR04, SZ05]. La méthode RANGER [PL06] propose d'explorer l'espace des preuves par résolution de manière stochastique. c'est une idée similaire à la méthode GUNSAT que nous proposons ci-dessous avec des différences que nous expliciterons. Les méthodes hybrides, quant à elles, font collaborer une méthode de recherche locale et une méthode systématique [MSG98, FH07, Gol08]. Elles permettent de prouver l'insatisfaisabilité, mais leur comportement est relativement différent des méthodes énoncées ci-dessus.

2.1 La méthode GUNSAT

Avec Laurent SIMON, nous avons proposé une méthode de recherche locale incomplète pour les problèmes d'insatisfaisabilité **[AS07b]** nommée GUNSAT. Cette méthode a pour une large part le même squelette que les méthodes de recherche locale classiques (voir algorithme 1.2) et est décrite dans l'algorithme 1.3. Elle se *ballade* au sein de l'espace de recherche par résolution, et essaye, à chaque étape, d'améliorer la preuve de résolution. Contrairement au cas SAT, il est difficile de savoir si une nouvelle clause obtenue par résolution améliore la preuve. Nous avons donc proposé une mesure qui approxime le nombre de modèles filtrés par chaque clause. Cette mesure considère les modèles filtrés par chaque couple de littéraux de la formule. Pour une formule contenant n littéraux, une clause filtre $2^{n-|c|}$ modèles.

Si l'on suppose que ces modèles filtrés sont supposés être également distribués sur les $|c| \times (|c|-1)/2$ couples de littéraux de la clause, on peut calculer le score d'un couple de littéraux $S(l_1, l_2)$, qui est la somme des modèles filtrés par les clauses où ce couple apparaît :

$$S(l_1, l_2) = \sum_{c \in \Sigma | (l_1, l_2) \in c} \frac{2^{n-1-|c|}}{|c| \times (|c|-1)}$$

Quand au score d'un couple de variables, nous l'avons défini comme étant la somme des poids au carré des 4 paires de littéraux associés : $S_q([x_1, x_2]) = S(x_1, x_2)^2 + S(x_1, \neg x_2)^2 + S(\neg x_1, x_2)^2 + S(\neg x_1, \neg x_2)^2$. Ainsi, plus le score d'un couple de variables est élevé, plus ce couple filtre de modèles et va permettre de prouver l'insatisfaisabilité de la formule. La fonction `ajouterClause` sélectionne le couple de variables ayant le plus haut score et essaye de générer une résolvante contenant un des 4 couples de littéraux associés, augmentant à nouveau son score. Si l'on échoue à générer une nouvelle clause avec ce couple de variables, on sélectionne alors le couple de variables avec le meilleur second score et ainsi de suite. Contrairement à l'analyse de conflit dans les méthodes CDCL, la nouvelle clause n'est générée que par une seule résolvante. L'algorithme 1.3 contient d'autres procédures. Nous allons les passer en revue.

La saturation des clauses binaires a été utilisée avec succès dans le pré-traitement des formules [Bac02]. Chaque fois qu'une nouvelle clause binaire est générée, la méthode `2-saturation` effectue toutes les résolutions possibles avec les autres clauses binaires. De plus, elle met en œuvre l'équivalence entre deux littéraux : si les deux clauses $(l_1 \vee \neg l_2)$ et $(\neg l_1 \vee l_2)$ sont générées, elle remplace toutes les occurrences de l_1 par l_2 (ou l'inverse). Cette saturation peut produire des clauses unitaires qui seront à leur tour propagées. C'est donc cette méthode qui va prouver l'insatisfaisabilité.

Comme nous l'avons vu dans la section 4, la résolution étendue est un système de preuves plus puissant que la résolution classique. C'est dans GUNSAT que nous avons utilisé pour la première fois ce système de preuve. Si plusieurs tentatives pour augmenter le score d'un couple de littéraux (l_1, l_2) échouent, nous augmentons artificiellement leur score en ajoutant le lemme $x \iff l_1 \vee l_2$. Cette simple règle, implantée dans la fonction `resolutionEtendue` a amélioré de manière impressionnante les performances de GUNSAT.

Avant les solveurs CDCL, les techniques de type *lookahead* étaient couramment utilisées pour réduire les arbres de recherches. La méthode `lookAhead` implante les algorithmes proposés par Daniel LE BERRE dans [LB01].

Afin d'éviter une explosion mémoire, nous réduisons la base de clauses de deux manières différentes. Tout d'abord, nous supprimons aléatoirement une clause dès lors que la formule utilisée possède un nombre de clauses pré-déterminé (fonction `supprimerClause`). De plus, après un certain nombre d'essais, nous enlevons toutes les clauses qui ont été ajoutées et qui ne sont pas binaires. Cela nous permet de visiter un autre espace de recherche (fonction `reduire`) et peut être vu comme la génération d'une nouvelle interprétation complète dans un algorithme de recherche locale pour SAT.

Les méthodes RANGER [PL06] et GUNSAT ont le même squelette. Mais alors que le premier essaie de générer le plus rapidement possible des nouvelles résolvantes, le second utilise un mécanisme de raisonnement plus puissant et une heuristique plus fine pour essayer de produire les meilleures clauses possibles. De plus, les techniques additionnelles que nous venons d'énoncer font de GUNSAT un démonstrateur plus efficace que RANGER.

Bien entendu, nous ne pouvons pas comparer les résultats de la méthode GUNSAT avec des solveurs de type CDCL. L'écart de performances entre les deux est énorme. La première raison incombe aux struc-

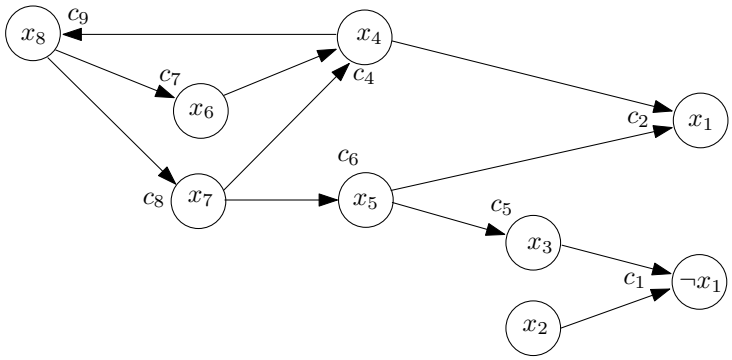


FIGURE 1.6 : *Graphe conflit obtenu associé à une interprétation complète*. On choisit la variable x_1 comme responsable du conflit.

tures de données utilisées dans GUNSAT qui sont quadratiques avec le nombre de variables. Le solveur GUNSAT est donc limité à des instances de petites tailles. De plus, à chaque conflit, les solveurs CDCL effectuent un ensemble de résolutions et apprennent un nogood qui a des propriétés bien particulières [PD08]. Ce n'est pas le cas de GUNSAT qui essaie lui de générer des résolvantes à priori intéressantes. Néanmoins, comparé à d'autres solveurs basés sur la résolution, tels que DR [DR94], ZRES [CS01] ou encore RANGER [PL06], notre solveur GUNSAT obtient de très bons résultats. Mais l'objectif de ce travail était tout autre. Tout d'abord, nous avons voulu répondre à un challenge proposé à la communauté [SKM97]. Nous avons également voulu proposer des méthodes alternatives pouvant aider à produire des preuves courtes mais aussi à mieux comprendre le problème SAT au détour de méthodes alternatives.

2.2 La méthode CDLS

Le sujet principal de la thèse de Jean-Marie LAGNIEZ, co-encadrée par Bertrand MAZURE, Lakhdar SAÏS et moi-même porte sur la recherche locale pour l'insatisfaisabilité. Elle est au coeur du projet ANR blanc UNLOC. Les premiers travaux que nous avons réalisés ensemble ont porté sur l'extension de l'analyse de conflit dans les algorithmes de type WSAT (voir algorithme 1.2) **[ALMS09a, ALMS09c]**. Cette extension présente des difficultés inhérentes aux méthodes de recherche locale. Celles-ci utilisent des interprétations complètes dans lesquelles plusieurs clauses peuvent être falsifiées. L'absence de notion de niveau de décision participe également à cette difficulté.

Nous avons donc commencé par définir ce qu'est un graphe conflit pour une interprétation complète. Comme auparavant, nous montrons sur un exemple un tel graphe conflit ; les définitions formelles sont accessibles dans les articles associés **[ALMS09a, ALMS09c]**.

On considère la formule Σ contenant les clauses suivantes :

(c_1)	$\neg x_1 \vee \neg x_2 \vee \neg x_3$	(c_2)	$x_1 \vee \neg x_4 \vee \neg x_5$	(c_3)	$x_2 \vee \neg x_1$
(c_4)	$x_4 \vee \neg x_7 \vee \neg x_6$	(c_5)	$x_3 \vee \neg x_5$	(c_6)	$x_5 \vee \neg x_7$
(c_7)	$x_6 \vee \neg x_8$	(c_8)	$x_7 \vee \neg x_8$	(c_9)	$x_8 \vee \neg x_4$
(c_{10})	$x_1 \vee \neg x_8$	(c_{11})	$x_7 \vee \neg x_9$		

On considère également l'interprétation complète $\mathcal{I}_c = \{x_1, x_2, \ldots, x_9\}$. La formule Σ est donc falsifiée. Le graphe conflit est construit en choisissant une clause falsifiée et en remontant, tant que cela est possible, les raisons (clauses unisatisfaites par l'interprétation complète). Ici, seule la clause c_1 est falsifiée, on choisit une variable de cette clause comme étant la cause du conflit, ici x_1, et on en

déduit le graphe conflit schématisé dans la figure 1.6. Différents choix sont possibles pour choisir la clause falsifiée et les clauses unitaires, et donc en règle générale, le graphe conflit n'est pas unique. Il faut également noter que le graphe conflit peut contenir des cycles, ce qui peut conduire à construire des résolvantes tautologiques. Enfin, l'existence d'un tel graphe n'est pas garantie. Nous avons démontré que ce graphe peut être construit dès lors que la clause falsifiée utilisée était critique [8]. Or, dans [GMP06], les auteurs ont prouvé que dans un minimum local toutes les clauses falsifiées sont critiques. Dès lors, nous construisons et utilisons le graphe conflit lorsque l'on se trouve dans un minimum local. Comme dans [CI96], nous avons donc utilisé le nogood généré par le graphe conflit pour sortir des minima locaux. Mais notre nogood est construit à l'aide de l'analyse de conflits et va donc être généralement basé sur plusieurs étapes de résolutions.

Même si l'existence d'un graphe conflit est assurée dans un minimum local, sa construction est difficile. Nous avons donc proposé une première approche basée sur la propagation unitaire. Partant de l'interprétation conflictuelle complète $\mathcal{I}_c$, nous construisons une interprétation partielle, l'interprétation partielle dérivée, dont les points de choix ont les mêmes valeurs dans les deux interprétations. La propagation unitaire sera naturellement faite sur les clauses unisatisfaites par l'interprétation complète. Cette interprétation partielle dérivée permet de définir un graphe d'implication classique et d'effectuer l'analyse de conflit (voir section 1.2). Le choix des variables de décision conduit à diverses interprétations partielles dérivées. Nous reprenons l'exemple ci-dessus et considérons deux interprétations partielles dérivées. Dans le cas où l'on considère l'ordre lexicographique pour choisir les variables de décision, on obtient :

$$\mathcal{I}_{pl} = \{\langle x_1@1, x_2@1, \neg x_3@1\rangle\}.$$

À ce niveau, les interprétations partielles dérivées et complètes diffèrent. On va donc terminer la construction en ajoutant artificiellement le littéral x_3 et en lui trouvant une raison de son affectation. On aura donc $\mathcal{I}_{pl} = \{\langle x_1@1, x_2@1, \neg x_3@1\rangle, \langle x_5@2, x_3@2\rangle\}$. D'un autre coté, si on tient compte de l'ordre lexicographique inverse, on obtient l'interprétation partielle dérivée :

$$\mathcal{I}_{pl_i} = \{\langle x_9@1, x_7@1, x_5@1, x_3@1\rangle, \langle x_8@2, x_6@2, x_4@2, x_1@2, x_2@2, \neg x_2@2\rangle\}$$

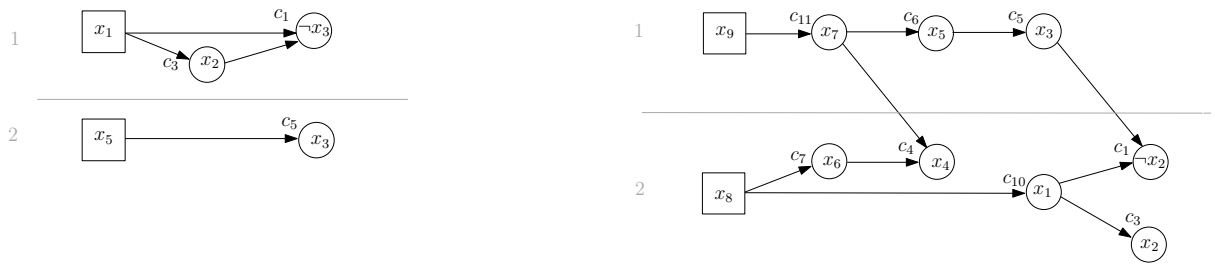
Dans les deux cas, les expressions partielles dérivées sont conflictuelles, les graphes d'implications qui s'en déduisent sont schématisés dans la figure 1.7. L'analyse de conflit classique peut être faite et le nogood résultant peut être ajouté à la base des clauses.

De plus, les interprétations partielles dérivées sont utilisées pour sortir du minimum local. Contrairement au démonstrateur WSAT classique qui n'autorise qu'un seul flip (changement de valeur de variables) à chaque étape, les variables flippées sont celles qui diffèrent de valeurs entre l'interprétation partielle dérivée et l'interprétation complète. On obtient ainsi une méthode à voisinage variable.

Les deux solveurs GUNSAT et CDLS sont deux solveurs de recherche locales capables de prouver l'insatisfaisabilité d'une formule. Néanmoins les ressemblances s'arrêtent là. En effet, GUNSAT est un démonstrateur qui explore l'espace des preuves par résolution de manière stochastique. A chaque étape, il essaie de trouver une résolvante qui semble la meilleure possible pour prouver l'incohérence. A contrario, le solveur CDLS est un solveur de recherche locale pour la satisfaisabilité. Il a le même squelette que l'algorithme 1.2. C'est l'ajout des nogoods, au travers de l'analyse de conflit, qui lui permet de prouver l'incohérence d'une formule.

Par contre, il est aussi important de noter que CDLS n'est pas un algorithme hybride comme [MSG98, FH07, Gol08]. C'est une méthode de recherche locale stochastique. La propagation unitaire est uniquement utilisée pour construire et analyser le graphe conflit et ainsi générer des nogoods.

8. Soit une formule CNF Σ, une clause c de Σ est critique pour l'interprétation complète $\mathcal{I}$ si et seulement si l'opposé de chaque littéral de c apparaît dans au moins une clause unisatisfaite par $\mathcal{I}$ [GMP06].



a. ordre lexicographique b. ordre lexicographique inverse

FIGURE 1.7 : *Graphe conflit construit à partir de l'interprétation partielle dérivée*. Comme précédemment, les nœuds carrés représentent les points de choix et les nœuds ronds, les littéraux affectés par propagation unitaire. La figure de gauche montre le graphe conflit lorsque l'ordre lexicographique est utilisé pour le choix des littéraux de décision. La figure de droite lorsque l'on utilise l'ordre lexicographique inverse.

Les travaux effectués autour de CDLS nous ont amené à proposer SATHYS, un démonstrateur *complet* hybride **[ALMS09b, ALM10, ALMS10]**. Nous ne détaillons pas cette méthode, mais la résumons rapidement. SATHYS utilise deux interprétations, l'une partielle $\mathcal{I}_p$ destinée au solveur de type CDCL qui essaie de prouver l'incohérence de la formule, et l'autre complète $\mathcal{I}_c$ destinée au solveur de recherche locale classique qui essaie de prouver la satisfaisabilité de la formule. A tout moment, $\mathcal{I}_p \subset \mathcal{I}_c$ et donc l'interprétation partielle sert de liste tabou dynamique à la recherche locale. Le solveur de recherche locale est le cœur de SATHYS et appelle le solveur CDCL lorsque les minimums locaux sont atteints. Les deux moteurs collaborent étroitement et échangent des informations utiles l'un pour l'autre. Le solveur SATHYS est aujourd'hui en plein développement et les résultats obtenus sont de plus en plus convaincants.

La table 1.2 résume sur les 3 catégories d'instances de la compétition SAT'09 les résultats de SATHYS, de solveurs hybrides (HINOTOS et HYBRIDGM) et des solveurs considérés comme faisant partie de l'état de l'art (PRECOSAT, GLUCOSE, CLASP et SATZILLA ont obtenu chacun une médaille d'or à la dernière compétition SAT). Notre solveur SATHYS obtient de biens meilleurs résultats que les autres solveurs hybrides. Il est moins bon mais compétitif comparativement aux solveurs CDCL (remarquer d'ailleurs que GLUCOSE, un des gagnants de cette compétition, résout moins d'instances quand le temps limite est réduit, il est par contre plus robuste). Le plus surprenant c'est que SATHYS est également plus robuste que SATZILLA qui est un solveur portefolio [XHHLB08]. Surtout, SATHYS est le solveur le plus robuste et il peut être considéré comme le premier vrai succès dans les méthodes de recherche hybride et une vraie réponse au challenge 7 proposé dans [SKM97, KS03].

		crafted		Applications		Aléatoires		
		SAT	UNSAT	SAT	UNSAT	SAT	UNSAT	total
SATHYS	**[ALMS09b]**	71	33	63	85	189	0	441
ADAPTG2	[LWZ07]	68	2	8	3	294	0	375
GNOVELTY	[Hoo02]	54	0	7	0	281	0	342
MINISAT	[ES04]	72	27	59	93	3	0	254
GLUCOSE	**[AS09b]**	75	39	54	98	17	0	266
PRECOSAT	[Bie08a]	81	41	65	99	2	0	288
CLASP	[GKNS]	78	53	53	85	66	18	353
HINOTOS	[LMS08]	69	36	39	68	65	11	288
HYBRIDGM	[BHG09]	51	5	5	0	294	0	350
SATZILLA_I	[XHHLB08]	86	42	60	82	90	55	415

TABLE 1.2 : Comparaison de différents solveurs sur les 3 catégories de la compétition SAT'09 : crafted (281 instances), applications (292 instances) et aléatoires (590 instances). Le temps est limité à 1200 secondes. Pour chaque solveur, on reporte le nombre d'instances SAT et UNSAT résolues.

D Travaux connexes

En marge de travaux autour de la résolution du problème SAT, nous avons également travaillé sur la représentation sous forme de graphe et sur un codage particulier de formules SAT. En effet, l'efficacité des algorithmes de résolution est dépendante de la représentation des problèmes utilisée. Le choix de la représentation est donc primordial pour permettre aux opérations les plus communes d'être réalisées le plus rapidement possible. Traditionnellement, les démonstrateurs SAT utilisent le codage sous forme CNF et la formule propositionnelle initiale est donc transformée en une formule CNF qui préserve la satisfaisabilité. Des variables supplémentaires sont généralement introduites pour éviter une explosion combinatoire de la taille de la formule. Néanmoins, cette transformation ne va pas sans inconvénients. Le codage sous forme CNF induit une perte d'informations structurelles utiles pour la résolution. Pour bénéficier de la structure du problème initial, diverses pistes ont été explorées. D'un coté, on peut essayer de déduire les propriétés au travers de la formule CNF. C'est le cas lors de la recherche de symétries [BS94, ARMS03], d'équivalences entre littéraux [Li03], de dépendances fonctionnelles [OGMS02, GMOS05, PTS07]. D'un autre coté, des recherches ont été effectuées pour utiliser un autre codage que la forme CNF, comme par exemple, des formules contenant des XOR [BM00], des circuits [JN00] mais également des formules plus générales [TBW04].

Par ailleurs, diverses représentations graphiques ont été élaborées pour représenter, modéliser et même résoudre des formules propositionnelles. Les diagrammes de décision binaires (BDD, voir également section 2.1) [Ake78, Bry86] peuvent être considérés comme une représentation graphique et permettent (au prix d'un effort calculatoire initial) de manipuler les formules propositionnelles. Dans [SD05], les auteurs proposent un outil de visualisation dynamique (au cours de la recherche) des instances CNF. La représentation graphique a également été proposée pour la décomposition [Dar04], pour représenter les formules 2-SAT [APT79]. On pourra enfin citer le graphe d'implications utilisé pour l'analyse de conflits [ZMMM01] ou pour simplifier les formules en pré-traitement [Bra01].

1 Codage basé sur les circuits

Avec Lakhdar SAÏS nous avons proposé un codage composé de portes et de clauses **[AS07a]** et un algorithme polynomial transformant toute formule CNF en ce un codage. Cet algorithme est basé sur certains travaux [Pur84, Bou96]. Ce codage peut ensuite être exploité pour extraire les ensembles *strong backdoor* ou pour effectuer des pré-traitements sur les formules. Nous commençons par introduire quelques nouvelles notions.

Etant donné une formule CNF Σ, elle peut être réécrite comme $\Sigma = (l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$. Où $\alpha(l) = \forall_{c \in \Sigma | l \in c} \cup \{c - \{l\}\}$ (respectivement $\alpha(\neg l) = \forall_{c \in \Sigma | \neg l \in c} \cup \{c - \{\neg l\}\}$) et $\Gamma = \{c | c \in \Sigma, c \cap \{l, \neg l\} = \emptyset\}$.

Exemple 1.18
Etant donnée la formule CNF $\Sigma = (x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee \neg x_2) \wedge (x_4 \vee \neg x_3)$. On a alors : $\alpha(x_1) = (x_2 \wedge (x_3 \vee \neg x_2))$.

Définition 1.19 une porte (booléenne) est une expression de la forme $y = f(x_1, \ldots, x_n)$ ou f est soit la conjonction ($\wedge$) soit la disjonction ($\vee$). La variable y est la variable de sortie et les variables x_i sont les variables d'entrée de la porte.

La porte $y = \wedge(x_1, \ldots, x_n)$ est représentée par l'ensemble de clauses $\{(y \vee \neg x_1 \vee \ldots \vee \neg x_n), (\neg y \vee x_1), \ldots, (\neg y \vee x_n)\}$. La porte $y = \vee(x_1, \ldots, x_n)$ est équivalente à la porte $\neg y = \wedge(\neg x_1, \ldots, \neg x_n)$. Une variable y est une variable de sortie d'un ensemble de portes $\mathcal{G}$ si elle est apparaît au moins une fois comme variable de sortie dans $\mathcal{G}$. Une variable x est une variable d'entrée de $\mathcal{G}$ si elle apparaît au moins une fois comme variable d'entrée dans $\mathcal{G}$ et jamais comme variable de sortie. On peut maintenant définir la notion de formule *circuit*-SAT.

Définition 1.20 Une formule *circuit*-SAT est une conjonction de portes $\mathcal{G}$ et de clauses C.

Pour construire les formules *circuit*-SAT nous utilisons les travaux issus de [Pur84, Bou96]. Dans [Pur84], Paul PURDOM a proposé un critère de branchement pour éviter de parcourir des interprétations redondantes. Alors que Yacine BOUFKHAD [Bou96] a proposé des notions de modèles restreints qui lui ont ensuite permis de déterminer une nouvelle borne inférieure théorique du seuil de transition des formules 3-SAT. Il a également utilisé cette notion pour rajouter des contraintes limitant les modèles d'une formule à ce type de modèle. Les contraintes supplémentaires ajoutées dans ces deux articles sont assez semblables et leur principal défaut est la nécessité de transformer une DNF (*disjonctive normal form*, soit une disjonction de conjonctions de littéraux) en un ensemble de clauses. Commençons par rappeler la propoistion de Yacine BOUFKHAD :

Proposition 1.21 ([Bou96]) *Soient Σ une formule* CNF *et l un littéral Σ. Σ est satisfaisable si et seulement si $\Sigma \wedge (l \vee \neg\alpha(\neg l))$ est satisfaisable.*

En introduisant des nouvelles variables, nous générons une formule *circuit*-SAT :

Proposition 1.22

1. *Chaque $\alpha(\neg l)$ est de la forme $(l_1 \wedge l_2 \ldots \wedge l_m \wedge c_1 \ldots \wedge c_k)$ où les c_i sont de clauses telles que $|c_i| > 1$. Soit y_i une variable supplémentaire représentant la clause c_i. Ainsi, $\alpha(\neg l)$ et l'ensemble des portes $\{l = \wedge(l_1, \ldots, l_m, y_1, \ldots, y_k), y_1 = \vee(c_1), \ldots, y_k = \vee(c_k)\}$ sont équivalentes pour* SAT.
2. *Soit $\Sigma = (l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$ une formule* CNF. *Σ est satisfaisable si et seulement si $(l = \alpha(\neg l)) \wedge (l \vee \alpha(l)) \wedge \Gamma$ est satisfaisable.*

ALGORITHME 1.4 : Algorithme *circuit*-SAT

Input : Σ une formule CNF et $\mathcal{I}_s$ un ensemble de littéraux cohérent
Output : $\mathcal{G}$ un ensemble de portes et $uncov(\Sigma, \mathcal{G})$ les clauses non couvertes

1 $\mathcal{G} = \emptyset$;
2 $uncov(\Sigma, \mathcal{G}) = \Sigma$;
3 **while** $(\mathcal{I}_s \cap \mathcal{L}(uncov(\Sigma, \mathcal{G})) \neq \emptyset)$ **do**
4 choisir $l \in \mathcal{I}_s$;
5 $varsEntree = \emptyset$; /* variables d'entrée de la porte */
6 **foreach** $c \in \alpha(\neg l)$ **do**
7 $uncov(\Sigma, \mathcal{G}) = uncov(\Sigma, \mathcal{G}) - \{c\}$;
8 **if** $(|c| = 2)$ **then**
9 $varsEntree = varsEntree \cup \{c\}$;
10 **else**
11 **if** $(\exists y_j \in \mathcal{G}$ *tel que* $y_j = \vee(c))$ **then**
12 $varsEntree = varsEntree \cup \{y_j\}$;
13 **else**
14 $\mathcal{G} = \mathcal{G} \cup \{y_k = \vee(c)\}$;
15 $varsEntree = varsEntree \cup \{y_k\}$;
16
17 $\mathcal{G} = \mathcal{G} \cup \{l = \wedge(varsEntree)\}$;
18 $\mathcal{I}_s = \mathcal{I}_s - \{l\}$;
19 **end**
20 **end**

La propriété 1.22.1 illustre comment $\alpha(\neg l)$ peut être transformée linéairement en un ensemble de portes en utilisant des variables supplémentaires. La propriété 1.22.2 illustre comment les contraintes supplémentaires ajoutées un utilisant la proposition 1.21 peuvent être codées à l'aide d'une porte. La formule résultante est donc une formule de type *circuit*-SAT. Cette propriété décrit une étape de notre codage. En effet, en considérant un ensemble cohérent de littéraux $\mathcal{I}_s$, notre codage va itérer la propriété précédente sur chaque littéral de $\mathcal{I}_s$. On en déduit l'algorithme 1.4 qui, prenant en entrée une formule CNF et un ensemble cohérent de littéraux $\mathcal{I}_s$, génère une formule *circuit*-SAT équivalente à la formule originale pour le problème de satisfaisabilité. L'ensemble des clauses non utilisées par cet algorithme (voir ligne 7) est noté $uncov(\Sigma, \mathcal{G})$. La condition de la ligne 3 permet d'éviter de couvrir (utiliser) les clauses plus d'une fois. La condition de la ligne 11 évite d'introduire des variables supplémentaires inutiles.

Exemple 1.23
On considère la formule Σ composée des clauses suivantes

$$(c_1)\quad x_1 \vee x_2 \vee \neg x_3 \qquad (c_2)\quad x_2 \vee \neg x_3 \vee x_4 \qquad (c_3)\quad x_2 \vee x_3 \vee \neg x_4$$
$$(c_4)\quad x_1 \vee \neg x_2 \vee x_3 \qquad (c_5)\quad \neg x_1 \vee x_4$$

On considère également l'ensemble $\mathcal{I}_p = \{\neg x_1, x_2, \neg x_3, \neg x_4\}$. La formule *circuit*-SAT résultante est intégralement recouverte ($uncov(\Sigma, \mathcal{G}) = \emptyset$) et l'ensemble de ses portes est :
- $\mathcal{G}_1 = \{\neg x_1 = \wedge(y_1, y_2), y_1 = \vee(x_2, \neg x_3), y_2 = \vee(\neg x_2, x_3)\}$.
- $\mathcal{G}_2 = \{x_2 = \vee(x_1, x_3)\}$.
- $\mathcal{G}_3 = \{\neg x_3 = \wedge(y_3, y_4), y_3 = \vee(\neg x_4, x_2), y_4 = \vee(x_1, \neg x_2)\}$.
- $\mathcal{G}_4 = \{\neg x_4 = \wedge(y_1, \neg x_1)\}$.

Nous avons prouvé que $\mathcal{I}_s$ est un modèle de Σ si et seulement si $uncov(\Sigma, \mathcal{G})$ est vide (c'est le cas de l'exemple précédent). Par conséquent, la recherche d'un *circuit*-SAT qui couvre entièrement la formule est intraitable dans le cas général. Nous montrons dans la section suivante les applications possibles au codage *circuit*-SAT.

1.1 Applications

Ensembles strong backdoors : Les formules *circuit*-SAT dépendent de l'ensemble $\mathcal{I}_s$ choisi. On peut alors prendre en compte les littéraux à utiliser ou les clauses à couvrir. Ainsi, nous avons prouvé que si l'on choisit $\mathcal{L}^-(\Sigma)$ comme ensemble de littéraux d'entrée de l'algorithme 1.4 alors la partie non couverte de la formule est une formule de Horn. Les variables d'entrée de $\mathcal{G}$ sont alors un *strong backdoor* de la formule initiale.

pré-traitement : A contrario, si on choisit comme ensemble $\mathcal{I}_s$ les littéraux dont le nombre d'occurrences est borné par une constante k, on peut utiliser *circuit*-SAT comme une technique de pré-traitement. Le pré-traitement des formules est utilisé avant la recherche afin de simplifier [SP05, EB05] ou de rajouter des contraintes aux formules [LA97, BS94, BW03, Bra01]. Le but étant d'obtenir une nouvelle formule, équivalente pour la satisfaisabilité, mais plus simple à résoudre. Étant donné une formule CNF Σ, notre technique de pré-traitement se décompose en deux étapes :

- Tout d'abord, nous générons une formule *circuit*-SAT à partir de l'algorithme 1.4. L'ensemble $\mathcal{I}_s$ utilisé est l'ensemble des littéraux dont le nombre d'occurrences est borné par une constante k.
- On transforme alors l'instance *circuit*-SAT en une nouvelle CNF.

2 Représentation graphique

Durant la thèse de Saïd JABBOUR, nous avons également travaillé sur une représentation graphique des formules SAT **[AJS07b, AJS08]**. Cette représentation peut être vue comme une extension du graphe d'implication utilisé pour représenter les formules 2-SAT [APT79]. Partant de cette représentation qui offre une vision alternative des formules SAT, nous avons proposé trois applications : la détection d'ensembles *strong backdoors*, la génération d'instances difficiles pour la résolution et une technique de pré-traitement.

Commençons par rappeler la représentation sous forme de graphe orienté pour les instances 2-SAT. Chaque variable de la formule est associée à deux nœuds du graphe, un pour chaque littéral. Chaque clause de la formule $(x \vee y)$ est associée aux deux arcs $(\neg x, y)$ et $(\neg y, x)$ décrivant les implications associées à la clause. L'algorithme polynomial utilisé pour déterminer la satisfaisabilité d'une instance 2-SAT est basé sur la fermeture transitive de ce graphe. Il suffit ensuite de vérifier qu'il existe un chemin menant d'un littéral à son opposé et vice versa pour prouver l'insatisfaisabilité de la formule. Il est clair que la fermeture transitive de ce graphe correspond à la saturation par résolution.

Nous proposons une extension de cette représentation aux formules SAT quelconques. Chaque arc de la clause possède un contexte qui est la condition nécessaire à l'implication.

Définition 1.24 Soit une clause c 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 $|\eta_c| = |c| - 2$.

Lorsque η_c est vrai, la clause c devient binaire. Une clause $c = l_1 \vee l_2 \vee \neg\eta_c$ peut donc être réécrite $((\neg l_1 \vee \eta_c) \rightarrow l_2)$ ou bien $((\neg l_2 \vee \eta_c) \rightarrow l_1)$. Chaque clause c possède $\frac{|c|\times(|c|-1)}{2}$ contextes et peut être

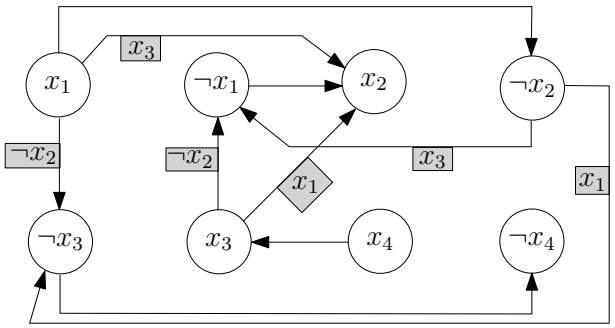


FIGURE 1.8 : *Graphe-*SAT *représentation*. Les littéraux sont représentés par deux nœuds. Les étiquettes (en gris) sont les contextes associés aux implications.

réécrite de $|c| \times (|c| - 1)$ façons différentes. A partir de là, on peut étendre le graphe d'implications pour 2-SAT aux formules quelconques.

Définition 1.25 Soit une formule CNF Σ, le graphe $\mathcal{G}_\Sigma = (S, E, v)$ associé à Σ est défini de la manière suivante :

- $S = \{x, \neg x$ tel que $x \in \mathcal{L}(\Sigma)\}$
- $E = \{a = (\neg x, y) | \exists c \in \Phi, c = (x \vee \alpha \vee y)$ où α est un sous-ensemble de $c\}$. Chaque arête $a \in A$ est étiqueté par l'ensemble $\overline{\alpha}$ i.e. $v(a) = \overline{\alpha}$
- Une clause unitaire l est associée à l'arête $(\neg l, l)$

On considère la formule CNF $\Sigma = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \neg x_3) \wedge (x_3 \vee \neg x_4)$ La représentation sous forme de graphe associée à Σ est schématisée dans la figure 1.8.

A l'aide de définitions supplémentaires, nous avons montré une relation entre la résolution classique et la fermeture transitive de ce graphe. Bien entendu, la fermeture transitive a une complexité exponentielle. Il est d'abord important de noter que dans la définition 1.25 chaque clause est représentée $|c| \times |c| - 1$ fois. Cette duplication des clauses n'est nécessaire que pour que la fermeture transitive soit complète pour la réfutation. Pour des formules de grande taille, cette représentation est irréalisable. Suivant l'utilisation qui en est faite, on peut représenter chaque clause par un seul et unique arc. Le graphe obtenu est tout de même équivalent à la formule originale pour le problème de satisfaisabilité. Mais dans ce cas, certaines résolvantes ne peuvent alors plus être obtenues.

Cette représentation offre une nouvelle vision des dépendances entre clauses et littéraux. De nombreux problèmes importants peuvent être formulés à partir de cette dernière : la recherche du plus court chemin entre un littéral et son opposé fournit la condition minimale pour laquelle ce littéral est impliqué. Nous présentons dans la section suivante des applications directes issues de cette représentation.

2.1 Applications

Calcul d'ensembles strong backdoors : La représentation sous forme de graphe étant une généralisation de la représentation pour 2-SAT, les ensembles *strong backdoors* que nous calculons sont donc évidemment considérés par rapport au fragment 2-SAT. Pour cela, nous n'avons besoin de représenter les clauses que par un et un seul arc. L'ensemble *strong backdoor* résultant est alors l'union des contextes de toutes les clauses. En effet, une fois ces variables affectées, les clauses sont satisfaites ou binaires. Néanmoins, notre but étant de construire le plus petit ensemble possible, nous devons choisir avec soin,

parmi les $|c| \times |c| - 1$ possibilités, les arcs représentant chaque clause. Le choix du contexte d'une clause donnée va donc prendre en compte les contextes des clauses déjà représentées. Le but est de minimiser la taille de l'union de tous les contextes.

Construire des formules difficiles : Nous avons utilisé la représentation graphique pour générer des instances SAT difficiles pour la résolution. Ces instances ont la particularité de posséder (au moins) deux littéraux impliqués. Mais la découverte de ces implications est difficile. Cette construction s'appuie sur l'application multiple de règles d'hyper-résolution.

Pré-traitement : Finalement, en utilisant une autre restriction à la représentation graphique, nous avons proposé une autre technique de pré-traitement des formules qui étend la notion d'hyper-résolution binaire initialement proposée dans [BW03].

Sommaire

Chapitre

2

Autour de QBF

LE PROBLÈME QBF (« *Quantified Boolean Formulas* » ou Formules Booléennes Quantifiées en français) est une extension naturelle du problème SAT. Les variables d'un problème QBF peuvent être quantifiées existentiellement ($\exists$), comme c'est le cas dans un problème SAT, mais aussi universellement ($\forall$). Cette simple modification n'est pas sans conséquence. Tout d'abord, en utilisant les variables quantifiées universellement, on représente de façon plus concise que dans SAT certains problèmes comme ceux issus de la vérification formelle ou de la planification. Cela permet également de représenter des problèmes qui ne peuvent pas l'être en SAT comme la planification dans l'incertain [Rin99a, FG00], le raisonnement dans les logiques des défauts [EETW00] ou modales [PV03]... Mais, et c'est le cœur du problème, l'ajout du quantificateur universel place le problème QBF plus haut dans la hiérarchie de la complexité que le problème SAT. En effet, le problème QBF a été le premier à être démontré comme appartenant à la classe PSPACE-complet [SM73].

Depuis une dizaine d'années, la résolution des problèmes QBF a subi un regain d'intérêt. Plusieurs raisons expliquent cela. Les instances SAT proposées sont de plus en plus grandes (jusqu'à 10 millions de variables et 30 millions de clauses), la compacité induite par la représentation en QBF peut donc s'avérer utile. Les énormes progrès réalisés sur SAT (voir le chapitre 1) ces dernières années ont aussi influencé la recherche sur les QBF. C'est pour cela que de nombreuses méthodes ou algorithmes proposés pour les QBF sont directement issus de ceux du problème SAT.

Ce chapitre est entièrement dédié à la résolution des problèmes QBF. La première section présente les notions nécessaires à la compréhension du reste du chapitre. La section B introduit les notions de symétries dans le QBF et propose des algorithmes pour les supprimer. La dernière section introduit un nouveau démonstrateur QBF qui s'affranchit de l'ordre habituellement imposé par les quantificateurs.

A Pré-requis

1 Définitions

Dans ce mémoire, nous nous limitons aux formules QBF bien particulières : les formules booléennes quantifiées polies prénexe et fermées.

Définition 2.1 Une formule QBF Φ sous forme polie prénexe et fermée est de la forme :

$$\Phi = Q_1 X_1 Q_2 X_2 \dots Q_n X_n \Sigma$$

Où $Q_1 X_1 Q_2 X_2 \dots Q_n X_n$ est le préfixe avec $Q_i \in \{\exists, \forall\}$, X_i un ensemble de variables et Σ est une CNF appelée la matrice. De plus, cette formule vérifie les propriétés suivantes :

- l'ensemble des variables est tel que $\forall i, j \in [1, \dots n]$ avec $i \neq j$, on a $X_i \cap X_j = \emptyset$;
- les quantificateurs sont tels que $Q_i \neq Q_{i+1}$ pour $i \in [1, \dots, n-1]$.

Chaque $Q_i X_i$ est appelé groupe de quantificateur ou quantificateur tout court. $Q_1 X_1$ est le groupe le plus externe et $Q_n X_n$ le plus interne. Une variable $x \in X_i$ telle que $Q_i = \exists$ (resp. $Q_i = \forall$) est appelée variable *existentielle* (resp. *universelle*). Son rang, noté $rang(x)$, est égal à i. De plus, on note $\mathcal{V}^{\forall}$ l'ensemble des variables universelles et $\mathcal{V}^{\exists}$ l'ensemble des variables existentielles. Ces notations sont naturellement étendues aux notions de littéraux.

La restriction proposée au début de cette section n'en est pas vraiment une, puisque toute formule QBF peut être transformée en une formule QBF vérifiant la définition 2.1. Mais cela ne va pas sans contrepartie. La structure interne de la formule originelle peut être perdue (que ce soit celle de la matrice ou celle du préfixe) et de nombreuses variables peuvent être ajoutées. Comme cela a été le cas pour SAT, il y a donc eu quelques travaux visant à retrouver cette structure [Ben05b], à travailler directement avec une représentation plus générale [ESW06, GNT07, Sté07] ou sur une formulation basée sur les circuits [GIB09]. Nous nous sommes tout de même limités à l'étude de formules QBF telles que définies dans la définition 2.1.

La propriété suivante permet de supprimer certains littéraux universels d'une formule sans en changer la nature.

Proposition 2.2 *Soient une* QBF $\Phi = Q_1 X_1 Q_2 X_2 \dots Q_n X_n \Sigma$ *et une clause* $c \in \Sigma$. *S'il existe une variable universelle* $x \in c$ *telle que pour toute variable existentielle* $y \in c$ *on a* $rang(y) < rang(x)$ *alors on peut supprimer* x *de la clause* c *en préservant l'équivalence de la formule originelle.*

Ainsi, on peut admettre que le quantificateur le plus interne d'une formule QBF est existentiel. Une formule QBF contenant une clause avec uniquement des littéraux universels est donc trivialement invalide. Cette propriété est utilisée lors de l'opération de résolution.

A cause de leur caractère PSPACE-complet, la *validité* (ou satisfaisabilité, quoique le premier terme soit plus souvent utilisé) des QBF est plus difficile à définir que dans le cas SAT. Un *certificat*, nommé encore *politique* est un ensemble de modèles propositionnels (il est en effet nécessaire d'exhiber deux modèles propositionnels par variable universelle) de la matrice satisfaisant certaines conditions [CMFL+06]. Avant de définir la notion de politique pour les QBF nous devons introduire quelques notations.

Soit une formule QBF Φ et $\mathcal{S}$ un ensemble d'interprétations sur les variables de Φ, (noté $\mathcal{V}(\Phi)$ comme dans le cas de formules propositionnelles). La *In-projection* (resp. *Out-projection*) de $\mathcal{S}$ sur un sous-ensemble de variables X de Φ ($X \subseteq \mathcal{V}(\Phi)$) restreint chaque interprétation de $\mathcal{S}$ aux littéraux de X (resp. $V(\Phi) \backslash X$) et est notée $\mathcal{S}[X]$ (resp. $\mathcal{S} \backslash X$). Cette définition peut se généraliser aux ensembles de littéraux.

L'ensemble des affectations possibles sur X est noté 2^X. On peut alors définir la notion de politique d'une formule QBF.

Définition 2.3 (Politique totale) Soient une formule QBF $\Phi = Q_1X_1Q_2X_2 \dots Q_nX_n\Sigma$ et un ensemble de modèles $\pi = \{m_1, \dots, m_n\}$ de sa matrice Σ. π est une politique totale de Φ si et seulement si
- $n = 0$ et $\Sigma = \top$
- Si $Q_1 = \forall$ alors $\pi[X_1] = 2^{X_1}$ et $\forall \mathcal{I}_1 \in 2^{X_1}$ $\pi \backslash \mathcal{I}_1$ est une politique totale de $Q_2X_2 \dots Q_nX_n\Sigma_{|\mathcal{I}_1}$
- Si $Q_1 = \exists$ alors $\pi[X_1] = \mathcal{I}_1$ et $\pi \backslash \mathcal{I}_1$ est une politique totale de $Q_2X_2 \dots Q_nX_n\Sigma_{|\mathcal{I}_1}$

Définition 2.4 Le problème QBF est le problème de décision qui consiste à déterminer si une formule booléenne quantifiée est valide ou non (et donc de savoir si elle admet une politique totale).

Exemple 2.5
On considère la formule QBF $\Phi = \exists x_5x_6 \forall x_2x_4 \exists x_1x_3\Sigma$, où $\Sigma = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee x_3) \wedge (x_5 \vee x_6)$.

Cette formule est valide et

$\pi = \{(\neg x_5, x_6, x_2, x_4, \neg x_1, x_3), (\neg x_5, x_6, x_2, \neg x_4, \neg x_1, x_3), (\neg x_5, x_6, \neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_5, x_6, \neg x_2, x_4, x_1, x_3)\}$ est une politique totale de Φ. On a donc :

$$\pi[\{x_5, x_6\}] = \{(\neg x_5, x_6)\}$$

$$\begin{aligned}\pi' &= \pi \backslash \{x_5, x_6\} \\ &= \{(x_2, x_4, \neg x_1, x_3), (x_2, \neg x_4, \neg x_1, x_3), (\neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_2, x_4, x_1, x_3)\}\end{aligned}$$

$$\pi'[\{x_2, x_4\}] = \{(\neg x_2, \neg x_4), (\neg x_2, x_4), (x_2, \neg x_4), (x_2, x_4)\} = 2^{\{x_2, x_4\}}$$

$$\pi' \backslash \{\neg x_2, \neg x_4\} = \{x_1, \neg x_3\}$$

Les politiques peuvent être représentées sous forme arborescente comme le montre la figure 2.1.

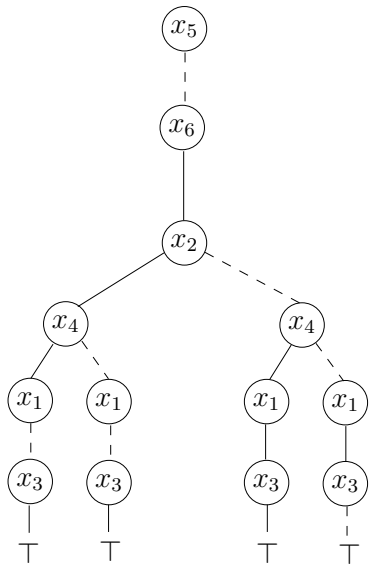


FIGURE 2.1 : *Représentation arborescente d'une politique totale d'une* QBF . Les cercles représentent les variables. Les traits pleins représentent leur interprétation à *vrai*, les pointillés à *faux*. Chaque variable existentielle a donc un fils et chaque universelle deux.

2 Démonstrateurs QBF

Le développement de démonstrateurs QBF a commencé au milieu des années 1990 [KBKF95, CGS98] et a reçu de plus en plus d'attention. Plusieurs raisons expliquent cela. les instances SAT proposées sont de plus en plus grandes (jusqu'à 10 millions de variables et 30 millions de clauses), la compacité induite par la représentation en QBF peut donc s'avérer utile. Les énormes progrès réalisés sur SAT (voir le chapitre 1) ces dernières années ont aussi influencé la recherche sur les QBF. C'est pour cela que de nombreuses méthodes ou algorithmes proposés pour les QBF sont directement issus de ceux du problème SAT.

Le préfixe d'une formule QBF impose deux contraintes. Tout d'abord, il est nécessaire de trouver un modèle propositionel pour chaque variable universelle augmentant ainsi l'espace de recherche à parcourir. De plus, le préfixe impose de suivre un ordre partiel dans le choix et l'élimination des variables.

Le premier solveur QBF proposé est une méthode adaptant les preuves par résolution aux formules QBF [KBKF95]. La Q-résolution est adaptée de la résolution propositionelle de la manière suivante : l'élimination des littéraux existentiels se fait en appliquant la résolution et en commençant par les littéraux les plus internes. L'élimination des littéraux universels se fait en appliquant la propriété 2.2. Une extension de ZRES [CS01] pour les QBF a également été proposée [PV04].

Les autres solveurs QBF se divisent en deux grandes catégories [9]. Dans la première, il y a les solveurs étendant les algorithmes de type DPLL [CGS98, Rin99b, Let02] et CDCL [GNT01a, GNT01b, ZM02]. Il faut noter que dans une QBF, une clause propagée unitairement n'est pas nécessairement de taille 1. Cela s'explique par la propriété 2.2. De plus, les heuristiques doivent choisir comme point de choix les littéraux appartenant au groupe de quantificateur le plus externe. Ceci entraîne un choix moindre et donc un espace de recherche potentiellement plus important. Des algorithmes de filtrage, comme la vérité et la fausseté triviale [CGS98] ou encore l'inversion des quantificateurs [Rin99b] ont été proposés. Contrairement au cas SAT, la propagation des littéraux purs (ceux n'apparaissant que positivement ou négativement dans la formule) est très importante pour l'efficacité des solveurs QBF [GNT04]. Le surcoût engendré par la mise à jour des littéraux purs est largement compensé par le fait que l'on doit trouver plus d'un modèle SAT.

Dans le cas de solveurs CDCL, l'analyse de conflits et l'apprentissage diffèrent également quelque peu du cas SAT. En effet, lors de l'analyse de conflit, on réitère le processus si le littéral assertif est universel et le *nogood* final est enregistré dans la base. Un autre type d'apprentissage peut être réalisé lors de la recherche : l'apprentissage basé sur les succès [ZM02, GNT06]. En effet, étant donné que l'on doit trouver un ensemble de modèles pour prouver la validité d'une QBF, on ne s'arrête pas au premier modèle trouvé. L'apprentissage basé sur les succès est donc le pendant de l'apprentissage basé sur les échecs, mais va enregistrer cette fois des *goods*, c'est à dire des conjonctions de littéraux. Ils autorisent un saut arrière non chronologique sur les littéraux universels. Les *nogoods* autorisent donc des coupures sur des interprétations connues pour être conflictuelles et les *goods* des coupures sur des interprétations connues pour être satisfaisables.

Dans la seconde catégorie de démonstrateurs QBF on trouve les solveurs qui éliminent les variables universelles pour revenir à un problème SAT. Ceci peut être réalisé par la résolution (en commençant par les variables existentielles les plus internes) [Bie04] ou en utilisant des fonctions de skolem [Ben04, Ben05a]. L'inconvénient majeur de ses deux méthodes est la taille de la CNF résultante qui peut croître exponentiellement. Malgré tout, elles obtiennent de bons résultats sur des classes de problèmes particuliers (voir les résultats de l'évaluation QBF proposée en marge de la conférence SAT

9. Igor STEPHAN propose une autre catégorisation des solveurs QBF tout aussi intéressante [Sté10].

`http://www.qbflib.org/index_eval.php`) . Espérant récupérer les points forts de ces deux approches, un démonstrateur combinant résolution et recherche a été proposé [PT09]. Finalement, comme pour SAT, des pré-traitements simplifiant la formule originale mais préservant l'équivalence existent [SDB06, MAVB10, EGN10].

B Suppression des symétries

De nombreux problèmes issus du monde réel contiennent des symétries, c'est-à-dire des permutations sur leur structure (par exemple leurs variables) les laissant invariants. Le problème des reines en est un exemple classique (rotation ou miroir du damier...). En tenant compte de ces régularités, on peut améliorer l'efficacité des démonstrateurs en évitant de parcourir des interprétations redondantes. C'est pour cela que les symétries ont été beaucoup étudiées dans le problème SAT [BS94, ARMS03] et les problèmes de satisfaction de contraintes [CJJ+06, LT09]. Avec Bertrand MAZURE, Saïd JABBOUR et Lakhdar SAÏS nous avons étudié la généralisation des symétries aux formules booléennes quantifiées **[AMS04a, AMS04b, AJS06, AJS07c, AJS07a]**. Dans un premier temps, nous avons défini les symétries pour les QBF et avons proposé une méthode de détection automatique de celles-ci. Nous avons ensuite proposé diverses approches pour supprimer les symétries. Tout d'abord, nous avons mis au point un solveur hybride composé d'un solveur QBF et d'un solveur SAT **[AMS04a, AMS04b]**. Nous avons également proposé deux techniques duales introduisant des nouvelles contraintes (SBP pour « *Symmetry Breaking Predicates* », voir [Cra92, Pug93, ARMS03, ARMS02]) dans la formule originale éliminant les interprétations isomorphes **[AJS06, AJS07a, AJS07c]**.

1 Introduction

Les symétries pour le problème SAT ont été introduites dans [Kri85] par KRISHNAMURTHY, où l'auteur améliore le système de preuves par résolution (voir section 4.1) en y ajoutant le principe des symétries. Mais c'est vers le début des années 1990 qu'une étude approfondie des symétries en logique propositionelle voit le jour [Cra92, BS94]. La définition des symétries pour les QBF est une généralisation de celle proposée dans [BS94] pour le problème SAT.

Définition 2.6 Soient $\Phi = Q_1 X_1 Q_2 X_2 \dots Q_n X_n \Sigma$ une formule booléenne quantifiée et σ une permutation des littéraux de Φ. Cette permutation peut être étendue à Φ et est une symétrie de Φ si elle vérifie :

1. $\forall l \in \mathcal{L}(\Sigma)\ \sigma(\neg l) = \neg\sigma(l)$
2. $\sigma(\Sigma) = \Sigma$
3. $\forall i \in \{1, \dots, n\}\ \sigma(X_i) = X_i$

Les deux premières conditions sont liées aux symétries dans le problème SAT. La dernière restreint les permutations entre les variables du même groupe de quantificateur. Une symétrie d'une formule QBF est donc aussi une symétrie de la matrice propositionnelle associée, l'inverse n'est pas vrai. Nous avons ensuite prouvé que :

Proposition 2.7 *Soient $\Phi = Q_k X_k, \dots, Q_1 X_1 \Sigma$ une formule booléenne quantifiée, σ une symétrie de Φ et π un ensemble d'affectations sur $\mathcal{V}(\Phi)$. π est une politique totale de Φ si et seulement si $\sigma(\pi)$ est une politique totale de Φ.*

Exemple 2.8
Soit $\Phi = \forall x_1 x_2 \exists x_3 x_4 \; \Sigma$ une QBF, où

$$\Sigma = \underbrace{(\neg x_1 \vee \neg x_2 \vee x_3)}_{c_1} \wedge \underbrace{(\neg x_1 \vee \neg x_2 \vee x_4)}_{c_2} \wedge \underbrace{(x_1 \vee \neg x_3 \vee \neg x_4)}_{c_3} \wedge \underbrace{(x_2 \vee \neg x_3 \vee \neg x_4)}_{c_4}$$

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 [10] de Σ. Dans la suite, les littéraux qui se permutent avec eux-mêmes sont omis (comme x_3 et x_4 dans σ_1). Φ est une formule valide et $\pi = \{(\neg x_1, \neg x_2, \neg x_3, \neg x_4), (\neg x_1, x_2, \neg x_3, x_4), (x_1, \neg x_2, x_3, \neg x_4), (x_1, x_2, x_3, x_4)\}$ est une politique totale de Φ. On peut vérifier que :

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

sont également des politiques totales de Φ.

Dans le contexte des formules propositionelles, les symétries peuvent être vues comme une relation d'équivalence entre les ensembles des interprétations de la formule booléenne, induisant des classes d'équivalence où chaque représentant est un modèle ou non. Il suffit de ne considérer qu'un seul représentant par classe pour avoir une méthode complète, réduisant d'autant l'espace de recherche. Les symétries sur les formules booléennes quantifiées étendent cette relation d'équivalence à des « super-ensembles » d'interprétations, chaque classe d'équivalence contenant ou non des politiques totales.

2 Détection

Il existe deux grandes familles pour détecter les symétries d'un problème à base de contraintes. La première utilise un algorithme dédié [BS94], la seconde réduit ce problème à un problème de contraintes particulier ou plus couramment à un problème d'automorphisme de graphes [ARMS03]. En effet, ils existe des algorithmes très efficaces pour résoudre le problème d'automorphisme de graphe [McK90, Soi93, KSM10]. C'est cette deuxième méthode que nous avons généralisée pour le cas QBF. La transformation d'une formule QBF en graphe est faite de la manière suivante :

- Chaque variable est représentée par deux sommets, un pour le littéral positif et l'autre pour le littéral négatif.
- On attribue une couleur pour chaque groupe de quantificateur. Les couples de littéraux d'un même groupe possèdent tous la même couleur (condition 3 de la définition 2.6)
- Chaque clause non binaire est représentée par un sommet de couleur blanc et une simple arête relie celui-ci à chaque littéral de la clause.
- Chaque clause binaire est représentée par une double arête reliant ses deux littéraux (gain en espace).
- Les sommets représentant un littéral et son opposé sont reliés par une simple arête (condition 1 de la définition 2.6).

La figure 2.2 montre le graphe associé à la formule Φ de l'exemple 2.8. Les automorphismes détectés par un programme comme SAUCY sont ensuite projetés sur les littéraux de la formule permettant de retrouver les deux symétries σ_1 et σ_2.

Comme dans le cas SAT, la suppression des symétries par ajout de contraintes (SBP) à la formule de départ se fait naturellement lorsque toutes les variables de la symétrie sont quantifiées existentiel-

10. Les symétries sont associées aux notions d'algèbre de groupes (symétriques) [RDO85]. Une telle représentation indique que $\sigma_1(x_1) = x_2$, $\sigma_1(x_2) = x_1$, x_3 et x_4 restant inchangés par σ_1. La composition de deux symétries est donc encore une symétrie.

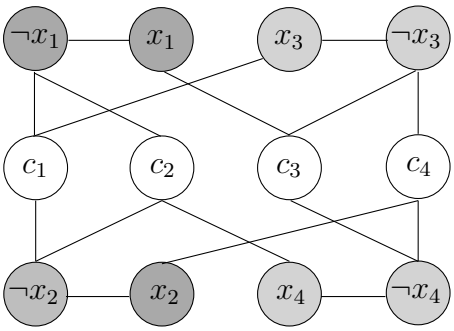


FIGURE 2.2 : *Détection des symétries dans une* QBF.

lement. Les SBP fixent un ordre sur les variables, et suppriment toutes les interprétations symétriques (redondantes) qui ne vérifient pas cet ordre. Ainsi, un seul représentant par classe d'équivalence des symétries va être parcouru durant la recherche, le plus petit. Considérons donc une symétrie $\sigma = (x_1, y_1)(x_2, y_2) \dots (x_n, y_n)$ où toutes les variables x_i et y_j sont existentiellement quantifiées. Les contraintes suivantes expriment alors l'élimination de la symétrie σ :

- $(x_1 \leq y_1)$
- $(x_1 = y_1) \rightarrow (x_2 \leq y_2)$

  $\dots$
- $(x_1 = y_1) \wedge (x_2 = y_2) \dots (x_{i-1} = y_{i-1}) \rightarrow (x_i \leq y_i)$

  $\dots$
- $(x_1 = y_1) \wedge (x_2 = y_2) \dots (x_{n-1} = y_{n-1}) \rightarrow (x_n \leq y_n)$

Lors de la transformation de ces contraintes en clauses, des variables supplémentaires sont ajoutées à la formule initiale pour éviter une explosion combinatoire. Malgré tout, des clauses de plus en plus grandes sont générées et dans la pratique on limite souvent le nombre de cycles pris en compte.

Par contre, dans le cas de symétries contenant des variables universelles, les prédicats cassant les symétries ne peuvent pas être ajoutés directement dans la formule initiale. Par exemple, la symétrie σ_1 de la formule Φ de l'exemple 2.8 porte sur les variables universelles x_1, x_2. Pour la supprimer, il est nécessaire d'ajouter le SBP $(\neg x_1 \vee x_2)$. Cette clause ne contenant que des variables universelles, elle rend la QBF résultante invalide alors que la QBF Φ originelle était valide. Nous introduisons par la suite divers travaux pour pallier ce problème.

3 Un démonstrateur hybride

Nous avons donc proposé une première approche où les prédicats SBP sont considérés comme une formule booléenne indépendante de la formule QBF initiale **[AMS04a, AMS04b]**. Un solveur hybride permet alors de traiter simultanément la formule SBP et la formule QBF.

Le démonstrateur hybride est constitué d'un démonstrateur QBF (basé sur DPLL ou CDCL) traitant la formule QBF initiale et d'un mécanisme de propagation unitaire sur une formule booléenne (le SBP). À chaque étape du solveur QBF, nous invoquons le démonstrateur SAT sur la formule SBP afin de propager les littéraux unitaires. Si x est la variable courante à affecter dans le solveur QBF, alors cette variable est également affectée dans le SBP à l'aide du solveur SAT. De manière réciproque, si y est un littéral déduit de la formule SBP, alors deux cas peuvent se présenter :

- Soit y est universellement quantifié dans la formule QBF, alors la branche correspondant à $\neg y$ est considérée comme vraie et le solveur QBF propage y pour affirmer cette hypothèse (si un modèle

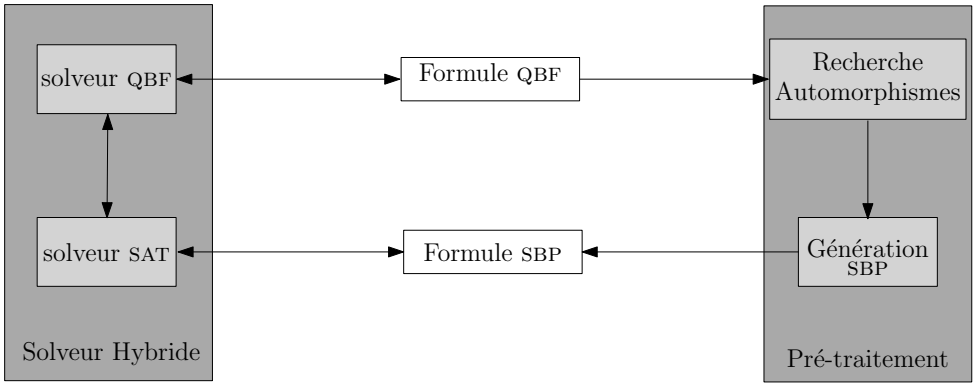


FIGURE 2.3 : Solveur QBF hybride cassant les symétries.

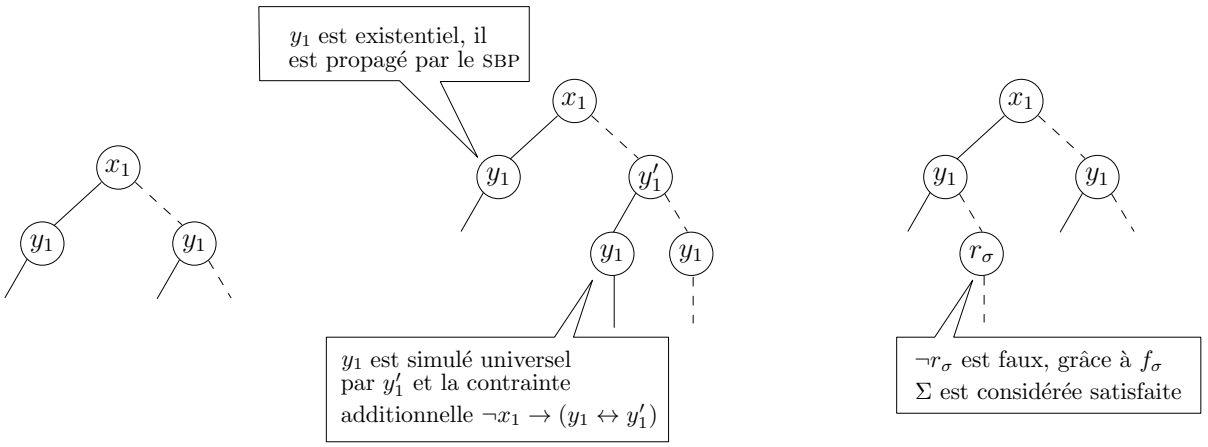
est trouvé avec y alors il existe un modèle avec $\neg y$, autrement un conflit apparaît puisque y est universellement quantifié).
- Soit y est existentiellement quantifié, alors la branche correspondant à $\neg y$ est considérée comme incohérente. Dans le même esprit, y est alors propagé dans le solveur QBF.

La figure 2.3 résume cette approche hybride. Le solveur QBF proposé prend une formule QBF en entrée. Une étape de pré-traitement est alors réalisée sur la formule QBF afin de déterminer les SBP, une version modifiée de SHATTER [ARMS03, ARMS02] effectue cette étape. Le solveur hybride, combinaison d'un solveur QBF et d'un mécanisme de propagation SAT (le solveur SAT considéré est en fait limité à un mécanisme permettant la propagation unitaire et les retours arrières, il ne prend aucune décision), opère alors sur la formule QBF originale et sur les SBP.

Habituellement, une fois les contraintes SBP ajoutées à la formule initiale, le nouveau problème peut être résolu par n'importe quel solveur existant ou à venir. Ce n'est pas le cas ici, et c'est l'inconvénient majeur de cette approche, puisqu'il est nécessaire de modifier le code des solveurs QBF que l'on souhaite utiliser. De plus, cette méthode s'applique uniquement à des démonstrateurs QBF effectuant une recherche arborescente comme SEMPROP [Let02], QUBE [GNT01a]... Pour pallier ce problème, nous avons proposé deux approches duales généralisant le principe des SBPs aux QBF.

4 Ajout de contraintes supplémentaires

Lorsque l'on génère le SBP, on doit considérer un ordre sur les variables de la formule. Dans le cas SAT, c'est l'ordre lexicographique qui est habituellement utilisé. Lorsque l'on veut supprimer les symétries dans une formule QBF, il faut que cet ordre tienne compte du préfixe. De plus, nous venons de voir que dans le cas des symétries contenant uniquement des variables existentielles, le SBP peut être ajouté naturellement à la formule initiale. Dans le cas de symétries contenant des littéraux universels nous avons proposé deux approches, l'une considère les interprétations symétriques que l'on veut éviter de parcourir comme cohérentes [AJS07a], l'autre comme incohérentes [AJS07c]. De par leurs aspects très techniques, nous ne rentrons pas dans les détails de ces deux approches, mais donnons une idée de leur fonctionnement en considérant une formule QBF $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Sigma$ et une symétrie $\sigma = (x_1, y_1)$ sur deux variables universelles. Les interprétations $\{x_1, \neg y_1\}$ et $\{\neg x_1, y_1\}$ sont donc symétriques. L'ordre $(x_1 \leq y_1)$ sur les variables nous permet de parcourir les interprétations contenant $\{\neg x_1, y_1\}$ et d'éviter les interprétations contenant $\{x_1, \neg y_1\}$.



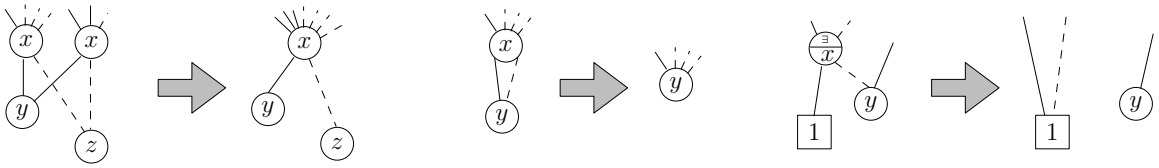
a. Comportement espéré b. Approche basée sur les nogoods c. Approche basée sur les modèles

FIGURE 2.4 : Suppression de symétries par ajout de contraintes supplémentaires. Les cercles représentent les variables. Les traits pleins représentent leur interprétation à *vrai*, les pointillés à *faux*. La figure a schématise le comportement espéré, à savoir la suppression de l'interprétation $\{x_1, \neg y_1\}$. La figure b schématise le comportement de l'approche basée sur les nogoods et la figure c, celle sur les modèles.

Approche basée sur les nogoods L'approche basée sur les nogoods va considérer la première interprétation comme incohérente. Pour cela, elle va ajouter le SBP associé à la symétrie σ à la formule initiale, choisir d'affecter x_1 avant y_1, requantifier y_1 comme un littéral existentiel, introduire une nouvelle variable universelle y_1' et introduire la contrainte $\neg x_1 \rightarrow (y_1 \leftrightarrow y_1')$. Le caractère universel de y_1 va être simulé à l'aide de y_1' et de cette contrainte (nommée QSBP). Le comportement associé à cette approche est schématisé dans la figure 2.4.b. Cette approche nécessite la requantification de certaines variables universelles, l'ajout de nouvelles variables universelles ainsi que de contraintes additionnelles (le SBP et le QSBP). Elle se généralise à un nombre quelconque de symétries universelles.

Approche basée sur les modèles Dans l'approche basée sur les modèles, on veut considérer un ensemble d'interprétations, noté $\mathcal{I}_\sigma$, comme des modèles de Σ, ici ce sont les interprétations contenant $\{x_1, \neg y_1\}$. Pour cela, on ajoute une variable existentielle r_σ associée à σ et on réécrit la formule Φ de la manière suivante : $\Phi = Q_k X_k, \ldots, Q_1 X_1, r_\sigma((\Sigma \vee \neg r_\sigma) \wedge (f_\sigma(r_\sigma, sbp(\sigma)))$. L'ensemble de contraintes f_σ force la variable r_σ à être fausse sur les interprétations de $\mathcal{I}_\sigma$. Les clauses $(\Sigma \vee \neg r_\sigma)$ deviennent vraies, permettant ainsi d'éviter de parcourir les interprétations symétriques. Comme précédemment, le comportement de cette approche, qui se généralise aux cas de plusieurs symétries, est schématisé dans la figure 2.4.c. L'inconvénient majeur de cette méthode est qu'elle augmente la taille de toutes les clauses de Σ (on ajoute r_σ à chaque clause), ce qui augmente artificiellement la difficulté de la QBF à résoudre.

Nous avons effectué des expérimentations sur de nombreuses instances et en règle générale, l'approche basée sur les nogoods a de meilleures performances que celles basée sur les modèles. Plus il y a de symétries sur les variables universelles, mieux cela est. Et dans le cas où les symétries ne concernent que les variables du quantificateur le plus interne, les symétries sont la plupart du temps inutiles (de par l'ordre de recherche imposé par les quantificateurs).



b. Elimination des sous-graphes isomorphes

a. Elimination des noeuds redondants

c. Réduction existentielle

FIGURE 2.5 : Règles de réduction d'un BDD. Les arcs pleins (resp. pointillés) représentent la variable affectée à *vrai* (resp *faux*).

C Un solveur basé sur les BDDs

En compagnie de Lakhdar SAÏS, nous avons proposé un nouveau type de démonstrateur QBF basé sur les Diagrammes de Décision Binaires (BDD) **[AS04, AS05b, AS05a]**. Le principal atout de cette approche est qu'elle permet de s'abstraire durant la recherche de l'ordre imposé par les quantificateurs.

1 Diagrammes de décision binaires

Les diagrammes de décision binaires (BDD) [Ake78, Bry86] sont associés à une formule booléenne. Ils représentent des formules propositionnelles sous forme de graphe acyclique. Les BDD offrent une représentation alternative des formules booléennes. Une fois construit, ils permettent de réaliser certaines opérations logiques très efficacement. Il ont largement été utilisés en vérification formelle [BCM+92], dans les ATMS [RDK87], mais également pour représenter des ensembles de clauses [Min93, CS01]. Un BDD est un graphe orienté acyclique avec une racine et deux nœuds terminaux, notés 1-terminal et 0-terminal. Tout nœud non terminal est associé à une variable de la formule et possède deux arcs sortants correspondant aux deux affectations possibles de cette variable. Un diagramme de décision binaire ordonné (OBDD) est un BDD tel que les variables apparaissent au plus une fois et dans un ordre fixe dans tout chemin du graphe. Un OBDD réduit (ROBDD) est un OBDD résultant de l'application répétée des règles suivantes :

1. Élimination des sous-graphes isomorphes (figure 2.5.a).
2. Élimination des nœuds pour lesquels les deux arcs sortants référencent le même nœud (figure 2.5.b).

Ne traitant que des ROBDD dans cette synthèse, nous les nommons par simplicité BDD. De plus, le BDD associé à une formule Σ est noté BDD(Σ).

2 Un nouvel opérateur de réduction

Nous avons proposé un nouvel opérateur de réduction des BDD, appelé opération de réduction existentielle. Comme le montre la figure 2.5.c, lorsqu'un nœud x est quantifié existentiellement et que l'un de ses arcs sortants référence le nœud 1-terminal alors toute référence à ce nœud x peut être simplement remplacée par une référence au nœud 1-terminal. Rappelons de plus que lorsqu'un nœud x est quantifié universellement et que ses deux arcs sortants référencent le nœud 1-terminal, le nœud x peut être éliminé par un opérateur de réduction classique (figure 2.5.b). Ces trois règles de réduction permettent de prouver la proposition suivante sur laquelle est basé notre démonstrateur QBDD(SAT).

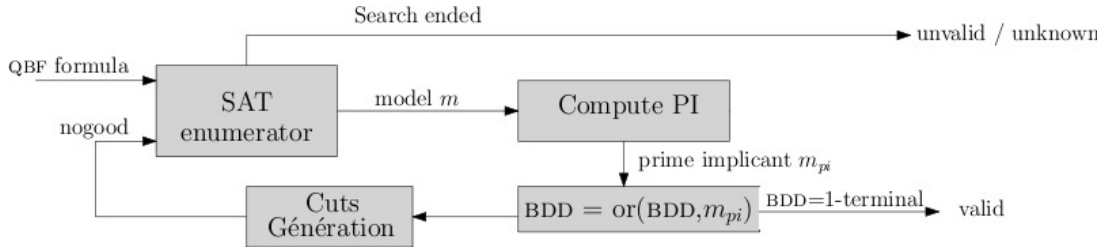


FIGURE 2.6 : architecture de QBDD(SAT)

Proposition 2.9 *Soient $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Sigma$ une formule booléenne quantifiée et $\pi = \{m_1, m_2, \ldots, m_n\}$ un ensemble de modèles de Φ. Si π est une politique totale de Φ alors par application des 3 règles 2.5.a, 2.5.b et 2.5.c,* BDD*(π) est réduit au nœud 1-terminal.*

3 La méthode QBDD(SAT)

Afin de libérer les solveurs QBF de l'ordre induit par le préfixe, nous proposons une combinaison des solveurs SAT classiques et des diagrammes de décision binaires, augmentés par notre nouvel opérateur de réduction défini dans la section précédente. La figure 2.6 donne l'architecture générale de notre approche symbolique QBDD(SAT). Pour vérifier la validité d'une formule QBF $\Phi = QX\Sigma$, notre approche utilise les démonstrateurs classiques du problème SAT pour rechercher les modèles de la formule booléenne Σ (*SAT enumerator*). Pour chaque modèle m trouvé, un impliquant premier, permettant de réduire l'espace de recherche, est extrait (*Compute PI*) et est ajouté disjonctivement au BDD (BDD = or(BDD, m_{pi})), BDD dont l'ordre est en accord avec l'ordre des variables dans le préfixe. Lorsque l'ensemble courant des impliquants premiers représente une politique totale, leur représentation sous forme de BDD est réduite au nœud 1-terminal (voir proposition 2.9) et le solveur QBDD(SAT) répond donc à la validité de Φ (valeur `valid`). Dualement, à la fin du processus de recherche, si le BDD n'est pas réduit au nœud 1-terminal, alors, et suivant la complétude de l'énumérateur SAT, notre approche QBDD(SAT) aura ou non prouvé l'invalidité de la formule.

Pour éviter de rechercher des modèles appartenant à des mêmes politiques potentielles, nous générons des coupures dans l'arbre de recherche, grâce à des *nogoods* injectés dans la formule de départ. Ces *nogoods* sont construits à partir des modèles et du BDD courant. Il est important de noter qu'ils n'ont rien en commun avec les *nogoods* obtenus par apprentissage des solveurs CDCL (voir section 1.2). Illustrons le principe en reprenant l'exemple 2.5. Supposons que le premier modèle généré par le solveur SAT soit $m = \{\neg x_1, x_2, x_3, \neg x_4, \neg x_5, x_6\}$. Un impliquant premier associé est $m_{pi} = \{x_2, x_3, x_6\}$. La contrainte ajoutée est $(\neg x_6 \vee \neg x_2)$. Elle évite la recherche de modèles inutiles sur la partie universelle déjà prouvée comme étant satisfaisable.

Comme nous l'avons précédemment mentionné, l'approche QBDD(SAT) peut être instanciée avec différentes techniques de recherche pour le problème SAT. Nous avons donc proposé une version basée sur un solveur de type CDCL (QBDD(DLL)) et une version basée sur un solveur de recherche locale pour la satisfaisabilité (QBDD(LS)). C'est à notre connaissance le seul solveur QBF incomplet basé sur une exploration stochastique de l'espace de recherche à avoir été proposé à ce jour. Les résultats expérimentaux de ces deux solveurs sont assez atypiques. Ils ont montré des résultats très intéressants sur des instances réputées difficiles pour les solveurs classiques.

Sommaire

Chapitre

3

Autour de SMT

COMME NOUS L'AVONS VU dans le chapitre 1, la syntaxe offerte par la logique propositionnelle est concise pour ne pas dire *pauvre*. C'est un atout, le problème SAT est très simple à comprendre et les démonstrateurs assez *faciles* à écrire (comparativement à des planificateurs ou des solveurs CSP). Mais cela peut également être un inconvénient majeur. En effet, certains problèmes pratiques sont plus facilement décrits et résolus en utilisant des logiques plus expressives. C'est le cas de problèmes issus d'automates temporels, d'ordonnancement (des équations mathématiques sont utiles), de vérification formelle concernant la mémoire des processeurs (des tableaux sont utiles). C'est cette richesse d'expressivité qu'offre la Satisfaisabilité Modulo Théories (SMT) [Sho79, GS96, PRSS99, AG93, ACG99]. Ainsi, un problème SMT contient des atomes propositionnels classiques, mais également des atomes issus d'une théorie donnée. Parmi ces théories, citons la théorie des logiques de différences, la théorie de l'arithmétique linéaire sur les rationnels ou les entiers, sur les vecteurs de bits. Les clauses obtenues peuvent donc être de la forme $\neg x \vee (v_1 = v_2 - 4) \vee lire(s, b - c) = d)$. On imagine alors facilement les avantages offerts par cette représentation. Certains chercheurs comme Robert NIEUWENHUIS y voient d'ailleurs une connexion entre les problèmes de satisfaction de contraintes (CSP) et les problèmes SAT [Nie06].

Ce chapitre est dédié à SMT et plus spécifiquement à SAT modulo la théorie des équations linéaires arithmétiques. Il est essentiellement issu de recherches effectuées en compagnie de Alessandro CIMATTI, Roberto SEBASTIANI et Arthur KORNILOWICZ lors d'un séjour post-doctoral en Italie (Trento). La section A introduit les définitions nécessaires à la compréhension du reste du chapitre ainsi que les deux types d'approches habituellement utilisées pour résoudre SMT. La section B traite de notre démonstrateur MATHSAT. La section C présente une application de MATHSAT.

A Pré-requis

1 Définitions

Nous commençons par introduire les différentes notions associées au problème SMT restreint aux équations linéaires sur les rationnels. Les formules résultantes sont nommées formules F_MATHSAT.

Définition 3.1 (atomes mathématiques) Un atome mathématique est construit au travers des constantes et variables rationnelles, et des opérateurs classiques de l'algèbre linéaire. Les atomes mathématiques ont donc la forme $(c_i \times v_i \bowtie c_k)$ ou $(c_i \times v_i \otimes c_j \times v_j \bowtie c_k)$ où les c_i sont des constantes rationnelles, les v_j sont des variables prenant également leurs valeurs sur $\mathbb{Q}$, $\otimes \in \{+, -\}$ et enfin $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$.

Définition 3.2 (formules F_MATHSAT) Étant donnés un ensemble de variables propositionnelles $x_1 \ldots x_n$ et un ensemble de variables rationnelles $v_1, \ldots, v_m$, une formule F_MATHSAT est une conjonction de clauses. Chaque clause est une disjonction d'atomes propositionnels (x_i ou $\neg x_i$) ou d'atomes mathématiques (y compris sous forme négative).

Exemple 3.3
La formule suivante est une formule F_MATHSAT :

$$\Sigma_{MS} = \begin{array}{ll} \{\neg(2v_2 - v_3 > 2) \vee x_1\} & \wedge \\ \{\neg x_2 \vee (2v_1 - 4v_5 > 3)\} & \wedge \\ \{(3v_1 - 2v_2 \leq 3) \vee x_2\} & \wedge \\ \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg x_1\} & \wedge \\ \{x_1 \vee (3v_1 - 2v_2 \leq 3)\} & \wedge \\ \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg x_1\} & \wedge \\ \{x_1 \vee (v_3 = 3v_5 + 4) \vee x_2\} & \end{array}$$

Intuitivement, une interprétation d'une formule F_MATHSAT est une extension d'une interprétation d'une formule propositionnelle classique et donne également une valeur (rationnelle) aux variables rationnelles. On peut donc définir ce qu'est un modèle pour une formule F_MATHSAT et ensuite définir un problème F_MATHSAT :

Définition 3.4 (modèle) Étant donné une formule F_MATHSAT Σ_{MS}, une interprétation $\mathcal{I}$ est un modèle de Σ_{MS} si elle satisfait chaque clause, soit par un atome propositionnel soit par un atome mathématique.

Définition 3.5 (problème F_MATHSAT) Le problème F_MATHSAT consiste à prouver si une formule F_MATHSAT possède ou non un modèle.

Exemple 3.6
Reprenons l'exemple 3.3. La formule Σ_{MS} est satisfaisable et l'interprétation $\mathcal{I}$ définie sur les variables rationnelles et propositionnelles vérifiant $\mathcal{I}(x_1) = false$, $\mathcal{I}(x_2) = false$, $\mathcal{I}(v_1) = 0$, $\mathcal{I}(v_2) = -1.5$, $\mathcal{I}(v_3) = -5$, $\mathcal{I}(v_5) = -3$ est un modèle Σ_{MS}. Notez que cette interprétation est partielle puisque v_4 n'a pas de valeur. Nous réécrivons ci-dessous la formule Σ_{MS} en soulignant les littéraux satisfaits par $\mathcal{I}$. Chaque clause de la formule est donc bien satisfaite.

$$\Sigma_{MS} = \begin{array}{ll} \{\neg(2v_2 - v_3 > 2) \vee x_1\} & \wedge \\ \{\neg x_2 \vee (2v_1 - 4v_5 > 3)\} & \wedge \\ \{(3v_1 - 2v_2 \leq 3) \vee x_2\} & \wedge \\ \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg x_1\} & \wedge \\ \{x_1 \vee (3v_1 - 2v_2 \leq 3)\} & \wedge \\ \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg x_1\} & \wedge \\ \{x_1 \vee (v_3 = 3v_5 + 4) \vee x_2\} & \end{array}$$

2 Démonstrateurs SMT

Les différentes approches habituellement utilisées pour résoudre le problème SMT se divisent en deux catégories. D'un coté, les approches dites pressées et de l'autre les approches dites paresseuses. Nous les présentons brièvement ci-dessous.

Dans les approches pressées, la formule F_MATHSAT est transformée en une formule SAT équivalente [BGV01, BV02, Str02]. L'avantage principal de cette approche est que la formule résultante peut être résolue par n'importe quel démonstrateur SAT. L'arrivée des solveurs CDCL a permis d'énormes avancées en utilisant cette façon de faire, notamment pour la vérification de processeurs avec le démonstrateur UCLID [LS04]. Il est tout de même obligatoire d'utiliser des algorithmes de codage très sophistiqués pour chaque type de théorie [BV02, PRSS99, MS05]. Malgré cela, l'inconvénient majeur de cette approche est la taille des formules SAT ainsi générées [dR04]. Les temps de transformation et de résolution par le solveur SAT choisi peuvent également être des obstacles majeurs à de telles approches.

A l'opposé des approches pressées, il y a les approches dites paresseuses. Celles-ci utilisent la force des solveurs spécialisés pour résoudre la théorie embarquée dans le problème SMT, mais également la force des solveurs SAT en résolvant une certaine formule booléenne [ACG99, dR02, NO05, Seb07]. Ces approches sont dites paresseuses car elles ne prennent pas en compte tous les atomes de la théorie T utilisée pour rechercher un modèle. Elles sont modulaires et flexibles mais ne tiennent pas compte de l'information donnée par la théorie pour guider la recherche. Dans la prochaine section, nous présentons MATHSAT, notre solveur paresseux pour résoudre les formules F_MATHSAT.

B La méthode MATHSAT

Ce solveur a été développé avec Alessandro CIMATTI, Roberto SEBASTIANI et Arthur KORNILOWICZ **[ABC+02b, ABC+02a]**. Il a depuis été réécrit [BBC+05, BCF+08], mais l'idée générale a été conservée.

1 Deux niveaux d'interprétations

Afin de pouvoir utiliser la technologie des solveurs SAT classiques, nous allons introduire une fonction bijective (appelée abstraction booléenne) faisant correspondre les atomes booléens à eux-mêmes et les atomes mathématiques à des nouvelles variables booléennes introduites pour l'occasion. Étant donnée une formule F_MATHSAT Σ_{MS}, nous notons $\mathtt{m2b}(\Sigma_{MS})$ la formule associée à cette bijection. C'est une formule propositionnelle classique. On peut donc faire abstraction de la partie mathématique et rechercher des modèles propositionnels qui satisfont $\mathtt{m2b}(\Sigma_{MS})$.

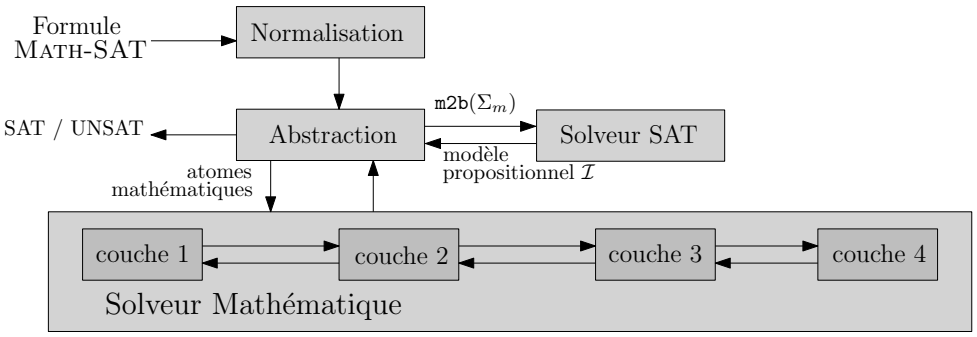


FIGURE 3.1 : Architecture du solveur MATHSAT

Reprenons l'exemple 3.3.

- L'interprétation propositionnelle $\mathcal{I} = \{\neg(2v_2 - v_3 > 2), \neg x_2, (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1), \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4)\}$ est un modèle propositionnel de $\texttt{m2b}(\Sigma_{MS})$; par soucis de lisibilité, nous n'avons pas introduit de nouvelles variables pour les atomes mathématiques, mais les considérons de cette manière dans l'interprétation $\mathcal{I}$, le littéral $(3v_1 - 2v_2 \leq 3)$ est donc vrai dans $\mathcal{I}$. Tout en étant un modèle propositionnel pour $\texttt{m2b}(\Sigma_{MS})$, $\mathcal{I}$ ne peut pas être prolongé en un modèle de Σ_{MS} car il n'existe pas de valeurs rationnelles de $v_1, \ldots, v_5$ vérifiant les atomes mathématiques se trouvant dans l'interprétation $\mathcal{I}$.
- L'interprétation $\mathcal{I} = \{\neg x_1, \neg x_2, \neg(2v_2 - v_3 > 2), (2v_1 - 4v_5 > 3), (3v_1 - 2v_2 \leq 3), (v_3 = 3v_5 + 4)\}$ est un modèle propositionnel de $\texttt{m2b}(\Sigma_{MS})$. Ce modèle peut être prolongé en un modèle de Σ_{MS}. En effet, en affectant $v_1 = 0$, $v_2 = -1.5$, $v_3 = -5$, et $v_5 = -3$, les atomes mathématiques de $\mathcal{I}$ sont satisfaits. Notez que dans ce cas, on retrouve le modèle exhibé dans l'exemple 3.3.

Il est clair que si la formule Σ_{MS} ne contient pas de modèles propositionnels de $\texttt{m2b}(\Sigma_{MS})$, alors la formule Σ_{MS} est insatisfaisable, mais que la détection d'un modèle propositionnel de $\texttt{m2b}(\Sigma_{MS})$ ne suffit pas à en déduire un modèle de Σ_{MS}. Le problème F_MATHSAT est donc au moins aussi difficile que le problème SAT.

2 Architecture

La figure 3.1 donne l'architecture générale du démonstrateur MATHSAT. La formule F_MATHSAT Σ_{MS} à résoudre subit d'abord une étape de normalisation. Nous reviendrons sur ce point par la suite. La couche d'abstraction gère la recherche. Un solveur SAT se charge de trouver des modèles propositionnels à $\texttt{m2b}(\Sigma_{MS})$. Une fois un tel modèle obtenu, la couche d'abstraction extrait les atomes mathématiques qu'elle fournit au solveur mathématique chargé de résoudre cette partie. Celui-ci se compose de plusieurs couches de complexité incrémentale, une couche d'un niveau n ne sera appelée que si les couches précédentes n'ont pas trouvé d'incohérence ou de modèles aux variables rationnelles. Le but est de trouver une solution au niveau le plus simple possible. Si la couche mathématique retourne un modèle alors la recherche est terminée, sinon, la couche d'abstraction demande un autre modèle propositionnel au démonstrateur SAT. Nous résumons les diverses couches mathématiques ci-dessous :

Couche 1 : Raisonnement sur les égalités Cette couche ne considère que les atomes mathématiques contenant des égalités, comme $v_1 = 3$ ou $v_1 + 2v_2 = 5$. Elle va créer des classes d'équivalence entre des variables rationnelles (l'équation $v_1 + 2v_2 = 5$ place les deux variables v_1 et v_2 dans la même classe d'équivalence). Chaque variable va ensuite être substituée par le représentant de sa classe. On

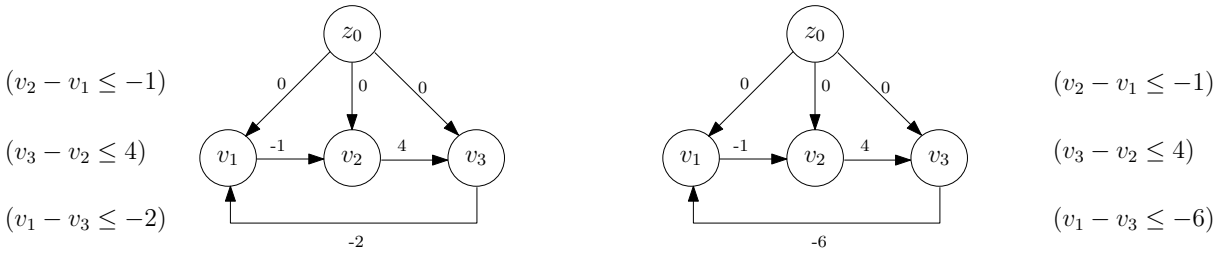


FIGURE 3.2 : Exemple d'utilisation de l'algorithme BELLMAN-FORD utilisé dans la couche 2 de MATHSAT. La figure de gauche montre un système de 3 équations. Chaque variable rationnelle est un sommet du graphe. Pour chaque équation $(v_i - v_j \leq c)$ on crée une arête de v_j vers v_i de poids c. On rajoute une variable imaginaire z_0 dont on fixe la valeur à 0. On ajoute pour chaque variable v_i une arête entre z_0 et v_i de poids 0 (équation $v_i - z_0 = 0$). On cherche alors les plus courts chemins entre z_0 et toutes les variables v_i. Celui-ci nous donne une solution du système d'équations. La figure de droite est un autre système d'équations, mais contenant cette fois-ci un cycle de poids négatif. Ce système n'a donc pas de solution.

supprime les affectations détectées ($v_i = 4$) et on les propage. Les atomes valides (comme $v_i - v_i \neq 2$ ou $v_i - vi > -1$) sont alors retirés. S'il existe un atome invalide (du style $v_i - v_i = 3$) alors on retourne au niveau de l'abstraction. Sinon, on passe à la couche suivante.

Couche 2 : Raisonnement sur des inéquations particulières Si tous les atomes mathématiques restants sont des inéquations du style $v_i - v_j \bowtie c$, avec $\bowtie \in \{<, >, \leq, \geq\}$, on peut utiliser un algorithme cherchant le plus petit chemin dans un graphe contenant des arcs pondérés négatifs avec détection de cycles négatifs. Nous avons utilisé une variante de l'algorithme de BELLMAN-FORD [CG99]. S'il existe un tel cycle alors une incohérence est détectée et on retourne au niveau de l'abstraction, sinon un modèle est obtenu. Un exemple d'utilisation de cette couche est montrée en figure 3.2. Cet algorithme est quadratique (par rapport au nombre de variables rationnelles) dans le pire des cas. Si cette couche ne peut pas être utilisée, on passe alors à la couche 3.

Couche 3 : Raisonnement général sur les inéquations Dans le cas où l'on n'a pas pu utiliser la couche 2 et qu'il n'existe pas d'inégalités ($v_i \neq v_j$), un algorithme de programmation linéaire comme le simplexe est appelé [Van96]. Un tel algorithme est exponentiel dans le pire des cas mais a souvent un comportement polynômial.

Couche 4 : Gestion des inégalités Les inégalités (du style $v_i \neq v_j$) ne peuvent pas être gérées par les couches précédentes. Le point positif est que de telles expressions n'existent que très rarement dans la pratique. Si tel n'est pas le cas, plusieurs solutions permettent de les gérer. La première d'entre elle consiste à transformer $(v_i \neq v_j)$ en $(v_i < v_j) \vee (v_i > v_j)$. Ceci est évidement inefficace. Nous avons donc choisi une autre solution. Nous commençons par ignorer toutes les inégalités et utilisons la couche 2 ou 3. Si le système d'équations résultant n'a pas de solutions alors, il n'y a aucun problème, on retourne au niveau de l'abstraction. Si le système résultant a une solution alors on vérifie que les inégalités sont respectées. Si ce n'est pas le cas, alors on divise ces inégalités en 2 ($(v_i < v_j) \vee (v_i > v_j)$) et on analyse les sous problèmes résultants. Heureusement ce dernier cas n'arrive que très rarement : une expression

$(vi \neq v_j)$ couvre une part de l'espace des solutions de mesure nulle alors qu'une solution d'un problème de programmation linéaire couvre un polyèdre qui contient une infinité de solutions.

3 Optimisations

Nous listons ci-dessous plusieurs techniques qui permettent d'améliorer MATHSAT. Certaines d'entre elles sont des adaptations pour les formules F_MATHSAT de travaux déjà existants pour les logiques modales [GS96, HPS98, GS00] ou la planification [WW99, ACG99].

Normalisation La figure 3.1 évoque une étape de normalisation effectuée avant la recherche. Cette étape est très importante. En effet, si deux atomes mathématiques sémantiquement équivalents mais syntaxiquement différents (comme $(v_1 \leq v_2)$ et $(v_1 - v_2 \leq 0)$) ne sont pas reconnus comme identiques, alors il peuvent être affectés différemment par le solveur SAT. L'espace de recherche se trouverait donc augmenté par des modèles propositionnels trivialement incohérents au niveau mathématique. A cette fin, l'étape de normalisation effectue les transformations syntaxiques suivantes :

- *Exploitation de l'associativité* : $(v_1 + (v_2 + v_3))$ et $((v_1 + v_2) + v_3)) \Longrightarrow (v_1 + v_2 + v_3)$.
- Tri des atomes :$(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1)$.
- *Factorisation* :$(v_1 + 2.0v_2 - 3.0v_3 \leq 4.0), (-2.0v_1 - 4.0v_2 + 6.0v_3 \geq -8.0), \Longrightarrow (0.25v_1 + 0.5v_2 - 0.75v_3 \leq 1.0)$.
- *Exploitation des négations* $(v_1 < v_2), (v_1 \geq v_2) \Longrightarrow (v_1 < v_2), \neg(v_1 < v_2)$.

Élagage au plus tôt Considérons une interprétation partielle $\mathcal{I}$ propositionnellement cohérente pour $\texttt{m2b}(\Sigma_{MS})$ mais mathématiquement incohérente. Alors toute interprétation $\mathcal{I}'$ telle que $\mathcal{I} \subset \mathcal{I}'$ est également mathématiquement incohérente. Nous avons donc ajouté une stratégie qui permet d'appeler les couches mathématiques de l'interprétation partielle générée par le solveur SAT. En cas d'incohérence, on peut alors faire un retour arrière dans l'arbre de recherche. Pour cause d'efficacité, cette stratégie ne peut être appelée à chaque nœud de l'arbre de recherche. On l'utilise par exemple quand de nouveaux atomes mathématiques ont été propagés (au niveau booléen) et que les variables rationnelles qu'ils contiennent sont déjà présentes dans l'interprétation partielle $\mathcal{I}$.

Propagation d'atomes mathématiques Supposons que la formule Σ_{MS} contienne les atomes mathématiques suivants : $(v_1 - v_4 \leq 4)$, $(v_1 - v_4 \leq 6)$ et $(v_1 - v_4 > 2)$. Supposons également que l'interprétation propositionnelle partielle $\mathcal{I}$ construite par le solveur SAT contienne le littéral $(v_1 - v_4 \leq 4)$, on peut alors propager dans le solveur SAT les littéraux $(v_1 - v_4 \leq 6)$ et $\neg(v_1 - v_4 > 2)$ et les ajouter dans l'interprétation $\mathcal{I}$. Cette technique et la précédente peuvent être réalisées en modifiant l'architecture de 3.1 afin que la couche d'abstraction et le solveur SAT communiquent plus étroitement : le solveur SAT doit informer la couche d'abstraction des littéraux qu'il propage, cette dernière pouvant à son tour lui fournir des littéraux à propager.

Saut arrière et apprentissage (mathématique) Comme cela est fait dans le cas booléen, on peut également faire du retour arrière non chronologique et de l'apprentissage au niveau des atomes mathématiques. En effet, si l'incohérence est détectée dans le solveur mathématique, il est alors possible d'en extraire un sous-ensemble (noté $\mathcal{I}_{cs}$) responsable de ce conflit et de l'utiliser au sein du solveur SAT. Pour cela, il faut backtracker au niveau de décision permettant de désaffecter l'atome le plus profond dans l'arbre de recherche. Par exemple, dans le cas de la couche 2, l'ensemble d'équations apparaissant

dans un cycle de poids négatif est responsable du conflit. Dans le cas de l'utilisation du simplexe (couche 3), il existe des stratégies permettant de déterminer l'ensemble conflit [WW99]. L'apprentissage consiste à ajouter à la formule Σ_{MS} la clause $\neg I_{cs}$.

Absorption Cette technique est un généralisation d'une technique proposée dans [WW99]. Si des atomes mathématiques n'apparaissent que positivement (resp. négativement) dans la formule Σ_{MS}, alors on peut supprimer toutes leurs occurrences négatives (resp. positives) dans les couches mathématiques. Cela est très intéressant, en particulier pour les égalités, puisque on évite de traiter les inégalités, limitant le nombre d'appels à la couche 4.

4 Conclusion

La méthode MATHSAT a été proposée afin de répondre à la validité des formules F_MATHSAT. Ces formules sont un cas particulier de SMT. Il est facile de comprendre maintenant l'architecture paresseuse des démonstrateurs SMT modulo une théorie quelconque $\mathcal{T}$. Il suffit, dans la figure 3.1 de remplacer l'appel au solveur mathématique par un solveur capable de résoudre des problèmes de $\mathcal{T}$. Des solveurs SMT gérant un plus grand nombre de théories ont donc été proposés, c'est le cas de la dernière version de MATHSAT [BCF+08], de BARCELOGIC [NO05]...

C Application à la vérification formelle temporelle bornée

En parallèle du développement de MATHSAT et toujours en compagnie de Alessandro CIMATTI, Roberto SEBASTIANI et Arthur KORNILOWICZ **[ACKS02, ABCS05]**, nous avons généralisé le «*Bounded Model Checking*» (ou encore vérification formelles bornée) [BCCZ99] aux automates temporels [Alu99]. La vérification de tels systèmes mettant en scène des automates à état fini contenant des horloges est un défi important pour la communauté. Habituellement, cette vérification passe par des matrices de différences bornées [Dil89]. Mais la quantité de mémoire nécessaire par ces approches est exponentielle en la taille des problèmes. D'autres techniques proposent une représentation symbolique de l'espace de recherche [Wan00, MLAH01].

Nous proposons de généraliser la vérification formelle bornée aux automates temporels. Les formules générées sont des formules F_MATHSAT qui peuvent donc être résolues par notre démonstrateur MATHSAT.

La vérification formelle d'un automate consiste à vérifier qu'une propriété, exprimée en logique temporelle LTL (« *Linear Temporal Logic* ») ou CTL («*Computational Tree Logic*») est valide. La logique LTL est une extension de la logique propositionnelle, qui contient en plus les opérateurs temporels suivants :

- L'opérateur G pour *Globally*. $G(\Sigma)$ signifie "à partir de maintenant et dans le futur" Σ doit être vraie.
- L'opérateur F pour *Future*. $F(\Sigma)$ signifie "maintenant ou dans le futur" Σ doit être vraie.
- L'opérateur X pour *neXt*. $X(\Sigma)$ signifie "au prochain point du temps" Σ doit être vraie.
- L'opérateur U pour *Until*. $(\Phi U \Sigma)$ signifie Φ doit être vraie jusqu'à ce que Σ soit vraie et Σ doit être vraie avant la fin du temps.

Plutôt que d'essayer de prouver qu'une propriété LTL est vraie, la vérification formelle bornée va essayer de montrer qu'elle est fausse (grâce à un contre-exemple) sur un nombre k borné d'étapes.

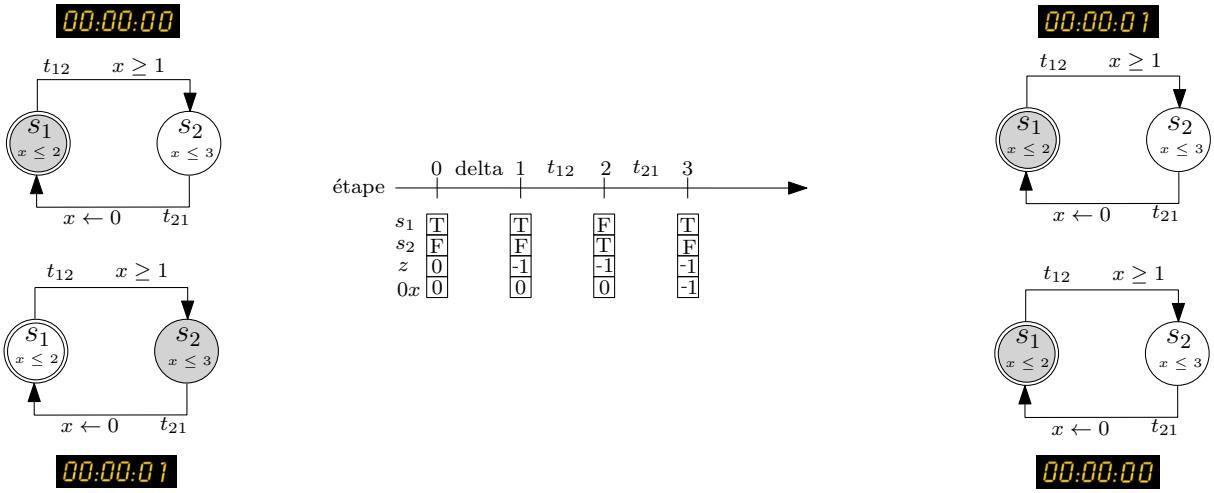


FIGURE 3.3 : Un exemple d'automate temporel avec diverses étapes possibles. Au départ on est dans l'état initial s_1 (état grisé). L'horloge x est initialisée à 0 (figure haut gauche). A chaque étape, on peut soit faire s'écouler le temps, soit passer d'un état à l'autre à l'aide des transitions (les étapes se lisent de gauche à droite, puis de haut en bas). Cela est modélisé par le schéma du milieu.

Il est alors possible de transformer ce problème en un problème SAT. Dans le cas où ce dernier est satisfaisable, alors un contre exemple de la propriété est exhibé (grâce au modèle obtenu). Dans le cas où il est insatisfaisable, on augmente la borne k permettant d'effectuer des étapes supplémentaires pour espérer découvrir le contre-exemple.

La généralisation aux automates temporels va se faire ainsi : chaque horloge (x) de l'automate est associée à une variable rationnelle ($0x$), on ajoute une variable z donnant le temps courant. La valeur d'une horloge (x) est obtenue avec $x = 0x - z$. En plus des étapes classiques faites dans les BMC, on ajoute une étape d'écoulement du temps qui est faite en diminuant la valeur de z. La remise à zéro d'une horloge se fait en lui donnant la valeur de z. Il est alors possible de transformer de tels problèmes en formules F_MATHSAT qui n'utilisent que les couches 1 et 2 du solveur MATHSAT (d'où l'intérêt d'avoir un algorithme ad-hoc pour cette dernière couche).

Nous donnons dans la figure 3.3 un exemple très simple d'automate temporel avec diverses étapes possibles. Cet automate possède deux états (s_1 et s_2). L'état initial est l'état s_1. On peut rester dans cet état tant que l'horloge x (symbolisée par le chronomètre) n'a pas dépassé deux unités de temps. La transition t_{12} permet de passer à l'état s_2. Pour passer à cet état, il faut que l'horloge x soit supérieure ou égale à une unité de temps. Lorsque l'on passe de s_2 vers s_1 on réinitialise l'horloge x. Nous montrons 4 étapes possibles de parcours de cet automate. À la première étape on fait avancer le temps d'une unité de temps (étape $delta$, la variable z est modifiée). A la deuxième étape, on passe à l'état s_2 (c'est possible car $0x - z \geq 1$). Enfin, à la dernière étape on retourne à l'état s_1. L'horloge x est réinitialisée (elle prend la valeur de z). Bien entendu, le codage en formule F_MATHSAT nécessite 4 variables pour s_1, une pour chaque étape (s_1^1, s_1^2, s_1^3, s_1^4). Il en est de même pour s_2, z et $0x$. Il est également nécessaire d'avoir des variables permettant de savoir quelle transition est effectuée à l'étape i (écoulement du temps, transition $t_{12}\ldots$).

Pour terminer, nous donnons dans la figure 3.4 un exemple un peu plus conséquent concernant plusieurs (seulement deux sont schématisés) processus essayant chacun d'atteindre une section critique (nommée cs) [Lam87]. La synchronisation se fait sur la variable id. Lorsque un processus i arrive dans l'état d'attente at, il affecte son identifiant à cette variable. Les autres processus peuvent également en-

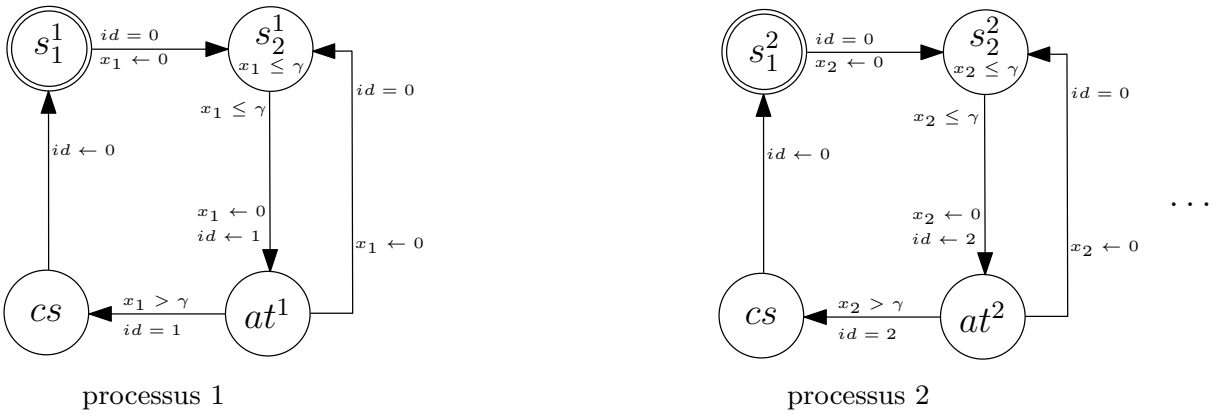


FIGURE 3.4 : Protocole d'exclusion mutuelle de FISCHER

trer dans cet état. Après un certain délai γ, si id est toujours égal à i alors ce processus peut entrer dans la section critique, sinon il retourne à l'état s_2^i, ce qui lui permettra d'essayer à nouveau d'arriver à la section critique. Ce protocole assez simple contient tout de même des notions intéressantes : écoulement du temps, synchronisation... Malgré sa simplicité, ce problème est difficile même avec peu de processus. Étant donnés n processus, nous avons considéré deux propriétés LTL à vérifier :

- *Accessibilité* : existe-t-il un état où tous les processus i sont dans l'état d'attente at^i. Ce qui se traduit par la propriété : $F(\wedge_i at^i)$. En utilisant la vérification formelle bornée, ce problème est insatisfaisable pour $k \leq n$ et satisfaisable autrement.
- *Équité* : Si un processus i se trouve une infinité de fois dans la section s_2^i alors il se trouve une infinité de fois dans la section critique. Nous cherchons un contre-exemple : $\neg(GFs_2^i \rightarrow GFcs)$. Cette propriété est insatisfaisable pour $k \leq n + 4$ et satisfaisable sinon.

Comparativement aux approches comme DDD [MLAH01] ou RED [Wan00], qui utilisent une représentation symbolique de l'espace de recherche, MATHSAT a obtenu des résultats bien meilleurs sur le premier type de problème testé. Quand au problème de l'*équité*, contrairement à MATHSAT, les démonstrateurs DDD et WANG00 ne gérant pas l'opérateur G, ils ne peuvent y répondre.

Conclusion et perspectives

DANS CETTE PREMIÈRE PARTIE, nous avons présenté une synthèse de nos travaux de recherche. Ils s'articulent autour des problèmes SAT, QBF et SMT, et plus particulièrement sur l'amélioration des techniques de résolution de ces trois problèmes. Le premier chapitre est consacré au problème SAT qui est le socle commun de nos travaux. Nous travaillons sur la résolution pratique de ce problème, en essayant de mieux comprendre le fonctionnement des solveurs de type CDCL, mais également sur le développement de nouveaux algorithmes incomplets pour prouver l'insatisfaisabilité. Nous nous intéressons aussi à différentes formes de représentation permettant d'extraire des informations utiles pour la résolution. Nous travaillons également sur deux types d'extensions du problème SAT. La première est le problème QBF, il est introduit dans le deuxième chapitre. Nous avons travaillé sur la suppression des symétries de ce problème et aussi sur un solveur qui s'abstrait, durant la recherche, de l'ordre imposé par les quantificateurs. Enfin, dans le troisième chapitre, nous avons présenté nos travaux réalisés sur une autre extension de SAT, le problème SMT, avec une méthode qui recherche des modèles propositionnels et essaient ensuite de les étendre dans la théorie donnée. Une extension du « *Bounded Model Checking* » aux automates temporels montre directement l'intérêt de ce formalisme. Tous ces travaux offrent des perspectives à plus ou moins long terme, nous les présentons maintenant.

Autour de SAT

L'étude des solveurs CDCL reste l'une de nos priorités. Même si les progrès réalisés par de tels démonstrateurs sont constants, ceux-ci sont toujours assez mal compris. Il en est pour preuve les différences de performances obtenues en changeant ne serait-ce qu'une de leurs constantes (poids VSIDS par exemple). Le rôle de chaque composant est encore assez méconnu, et que dire des interactions entre eux. Une des étapes importantes pour mieux comprendre ces démonstrateurs passe par une réelle étude expérimentale [Hoo94]. Avec Laurent SIMON nous avons déjà entamé cette étude et les premiers résultats ont donné lieu au solveur GLUCOSE qui fait aujourd'hui parti de l'état de l'art des solveurs. Nous continuons toujours dans cette direction et portons plus particulièrement notre attention sur l'impact des heuristiques dynamiques dans la recherche mais également sur le rôle des redémarrages et leur interactions avec ces heuristiques. Ces dernières récompensent les variables les plus conflictuelles. Souhaite-t-on alors que l'opération de résolution s'opère en priorité sur elles ? Si tel est le cas, il pourrait y avoir un antagonisme entre heuristiques et redémarrages. En effet, ces variables seront affectées en haut de l'arbre de recherche au prochain redémarrage, les empêchant assez sûrement d'être utilisées dans les processus de résolution.

A coté de cela, nous essayons de comprendre l'impact des clauses apprises dans le processus de recherche. Il semble que certaines clauses apprises ne soient d'aucune utilité, si ce n'est pour effectuer le saut arrière. S'il est possible de déterminer à l'avance de quelles clauses apprises le démonstrateur a besoin, il deviendrait possible de forcer leur création grâce à certaines résolutions. Cela pourrait s'avérer très bénéfique dans le cadre des architectures multi-cœurs qui deviennent aujourd'hui incontournables. Un cœur serait chargé de la recherche (le maître) et les autres cœurs seraient appelés afin de générer les bonnes clauses (les esclaves). Le maître peut alors continuer sa recherche comme si les clauses qu'il demande de trouver aux esclaves existaient réellement.

La proposition de nouvelles politiques de redémarrage dynamiques constitue également une voie de recherche. Les propositions faites jusqu'à maintenant ont permis d'améliorer les démonstrateurs, mais de nombreux progrès restent à faire. Nous comptons explorer plusieurs pistes. Premièrement, nous souhaitons proposer une politique de redémarrage dynamique pour notre solveur hybride SATHYS, en tenant compte des informations recueillies par le moteur de recherche locale. Nous souhaitons de plus, prendre en compte certaines caractéristiques de la formule, comme la présence de XOR, pour générer de nouvelles politique de démarrage.

Comme cela a été mentionné dans le premier chapitre, il est prouvé que les solveurs de type CDCL sont des systèmes de preuves aussi efficaces que le système de preuve par résolution. L'amélioration des démonstrateurs SAT peut donc passer par un système de preuve plus puissant que la résolution. Nous avons proposé une première approche visant à intégrer dans un démonstrateur CDCL une restriction de la résolution étendue (nommée LER). Une des premières perspectives de recherche à court terme consiste d'ailleurs à déterminer si LER est aussi puissant que la résolution étendue générale ou non. Nous souhaitons également réfléchir à d'autres systèmes plus puissants que la résolution et à leur intégration dans les démonstrateurs SAT. Une des solutions possibles est l'utilisation des symétries locales qui sont des symétries apparaissant dans des sous-problèmes durant la recherche. Qui plus est, nous pensons qu'il y a un lien entre la restriction que nous avons proposée (LER) et les symétries locales. Si tel est le cas, la connaissance des symétries locales nous donnerait les paires de littéraux candidates à la résolution étendue. Au moyen d'une représentation particulière des clauses sous forme de graphe, nous souhaitons également trouver quelles sont les variables importantes pour créer un pré-traitement qui ajoute les variables étendues avant la recherche.

Nous souhaitons également continuer nos recherches sur les méthodes incomplètes pour UNSAT. D'ailleurs, nous réfléchissons actuellement à l'amélioration de notre solveur CDLS. La première version de ce démonstrateur de recherche locale pour l'insatisfaisabilité utilise une reconstruction d'une interprétation partielle par propagation unitaire afin de pouvoir générer les clauses assertives. Nous voulons éviter cette étape, présente uniquement pour générer le nogood, à l'aide de clauses particulières, les clauses critiques. Nous travaillons également sur un Horn renommage des formules pour générer d'autres types de résolvantes supprimant par la même occasion la notion de clauses assertives. En effet, les clauses apparaissant dans un graphe conflit sont horn-renommables (cette réécriture est donnée par la polarité des littéraux dans le graphe d'implications). Une autre possibilité de méthodes de recherche locale consiste à ne considérer qu'une sous-formule de la formule initiale afin de ne tenir compte que d'un et un seul MUS (« *Minimum Unsatisfiable Core* » ou encore Noyau insatisfaisable minimum). Cela peut paraître étrange de ne considérer qu'une sous-formule, puisque plusieurs techniques de pré-traitement consistent au contraire à rajouter de nombreuses clauses (résolvantes). Néanmoins, il est étrange de remarquer que dans [GMP06], les auteurs proposent une méthode d'extraction de MUS qui, sur certaines instances, s'avère plus rapide que des démonstrateurs CDCL ne devant, eux, *que* prouver l'inconsistance de la formule. Certaines clauses *brouillent* donc les démonstrateurs et les supprimer accélérerait peut-être la recherche. Une telle méthode serait clairement incomplète. En effet, on ne peut rien déduire dans le

cas où la sous-formule est montrée satisfaisable.

Autour de QBF

Nous comptons orienter nos recherches autour des QBF sur deux axes différents. Le premier porte sur les symétries et plus précisément sur une autre forme de symétries. En effet, nous avons défini les symétries d'un tel problème avec une contrainte sur les variables : elles ne peuvent être symétriques qu'avec des variables du même groupe de quantificateur. Cette restriction permet de définir des symétries syntaxiques, elles laissent inchangée la formule initiale modulo une réécriture des clauses et des variables. Nous pensons supprimer cette restriction : des variables pourraient donc être symétriques avec celles d'un autre groupe de quantificateur. Une telle généralisation augmenterait le nombre de symétries possibles et permettrait de réduire d'autant l'espace de recherche. Ce n'est pas la définition en elle-même qui est compliquée mais l'utilisation qui pourrait en être faite. Une autre possibilité consiste à définir les concepts de symétries locales au sein des QBF et à les exploiter durant la recherche comme cela a été fait dans le cadre propositionnel [BS94].

A côté de cela, nous souhaitons également travailler au delà des formes prénexes fermées et polies. Le respect de l'ordre imposé par les groupes de quantificateurs est en effet très dommageable à la résolution. Cela a été mis en évidence dans le cadre des CSP quantifiés [BLV07] et dans le cadre QBF [AGS05]. Considérons par exemple un problème de vérification formelle bornée. Dans le cadre SAT, on est obligé de dupliquer la matrice de transitions de l'automate autant de fois que le nombre d'étapes voulues. Dans le cadre QBF, elle n'apparaît qu'une seule fois, les quantificateurs universels évitant la duplication. Néanmoins dans le cadre SAT on va pouvoir focaliser la recherche n'importe où, y compris sur la propriété LTL à vérifier. Dans le cas où l'insatisfaisabilité se situe dans cette région, elle peut être trouvée rapidement. Dans le cas QBF, on est obligé de commencer par l'état initial et on doit simuler le parcours de tout l'automate, rendant difficile la découverte de l'invalidité de la formule. Nous souhaitons donc aller au delà de cette représentation. Des travaux existent déjà [Sté06, Sté07, WKC10] mais nous pensons explorer d'autres voies. Malgré tout, il y a actuellement un manque criant d'instances QBF dans un format autre que celui énoncé ci-dessus. Nous voulons donc en construire, que ce soit des formules de vérification formelle bornée, ou encore de planification dans l'incertain. Cela nécessite donc la mise en œuvre de nouveaux codages et de nouveaux démonstrateurs.

Autour de SMT

Étant donné que nos recherches sur SMT sont en pause depuis un certain nombre d'années, il nous est plus difficile de nous projeter dans le futur. Néanmoins, il est une voie que nous souhaitons explorer. En effet, il est parfois admis que SMT est en quelque sorte une connexion entre SAT et CSP [Nie06]. Nous souhaitons alors pouvoir intégrer les contraintes globales au sein de SMT. Dans le même sens, nous souhaitons regarder s'il est possible d'ajouter la notion de contraintes plus globales au problème SAT. Une telle extension permettrait alors de gérer le codage des *1 parmi n*, souvent utilisé dans les codages vers SAT. On peut penser que les contraintes de cardinalité gèrent cela, mais grâce aux contraintes globales on peut obtenir une expressivité beaucoup plus élevée. Cette perspective de recherche se veut être à plus long terme.

Et après

Au delà des perspectives de recherche proposées dans les trois grands axes sur lesquels nous avons travaillé ces dernières années, nous souhaitons également explorer des nouveaux domaines.

Le premier d'entre eux est l'utilisation de SAT, y compris certaines extensions, pour traiter des problèmes de cryptographie asymétrique. Jusqu'ici SAT a été utilisé dans la cryptographie symétrique [MM00, SNC09]. En effet, des algorithmes de cryptage comme le DES (« *Data Encryption System* ») ou le MD5 utilisent des bits, décalages et des opérations XOR qui se codent naturellement en logique propositionnelle. De notre coté, nous souhaitons étudier s'il est possible d'utiliser les technologies SAT ou SMT dans les problèmes de cryptographie asymétrique. On a alors affaire à des problèmes de factorisation des grands entiers ou encore à des problèmes de logarithmes discrets. En collaboration avec des chercheurs issus du milieu de l'arithmétique des ordinateurs, nous essayons de monter un projet sur ce thème, offrant une perspective de recherche à plus long terme. Plus généralement, le coté applicatif, utiliser SAT dans des domaines bien précis, est une voie de recherche que nous comptons explorer.

Le problème SAT est un problème de décision. A ce titre, il ne fournit que peu d'informations dans le cas d'instances insatisfaisables. De nombreux problèmes réels, comme la création d'emplois du temps (« *timetabling* »), sont sur-contraints et n'acceptent pas de solution. Il est alors nécessaire de fournir des affectations satisfaisant le plus de contraintes (clauses) possibles. C'est le but du problème d'optimisation MAXSAT et de ses dérivées MAXSAT pondéré et MAXSAT partiel [LM09]. Dans le premier cas, on rajoute un poids à chaque clause et on souhaite maximiser le poids des clauses satisfaites, dans le second, il existe des clauses dures qui doivent absolument être satisfaites et des clauses souples qui peuvent être violées. Actuellement, il existe deux grandes catégories de méthodes de résolution de MAXSAT : les méthodes approximatives (comme les méthodes de recherche locale) et les méthodes exactes qui sont pour la plupart basées sur le principe de séparation et évaluation (« *Branch and Bound* »). Nous souhaitons explorer des nouvelles voies pour résoudre le problème MAXSAT. En particulier, la recherche hybride mêlant solveurs CDCL et recherche locale semble s'avérer une bonne alternative [KSGS09]. Les recherches et le développement de notre solveur SATHYS peuvent donc servir de base à de nouvelles techniques de résolution de MAXSAT.

Partie II

Curriculum vitae

Sommaire

Notice Individuelle

État civil

Gilles AUDEMARD

Né le 12 mai 1972 à Aix-en-Provence (Bouches-du-Rhône, France)

Nationalité française, pacsé, 3 enfants

Fonction actuelle

Maître de Conférences Classe Normale (section CNU 27 informatique)

Lieu d'exercice : **I**nstitut **U**nivertaire **T**echnologique de Lens (Université d'Artois)

Laboratoire de rattachement : **C**entre de **R**echerche en **I**nformatique de **L**ens (UMR 8188)

Coordonnées professionnelles

CRIL-CNRS UMR8188
Institut Universitaire Technologique
rue de l'université, SP18
F-62307 Lens

+33 (0)3 21 79 32 77

+33 (0)3 21 79 17 60

gilles.audemard@cril.fr

http://www.cril.fr/~audemard

Parcours professionnel

2002– ... **Maître de Conférences** à l'IUT de Lens. Bénéficiaire d'une PEDR depuis octobre 2008.

2001–2002 **Chercheur invité** à l'IRST, Trento (Italie).

1998–2001 **Allocataire de Recherche** (MESRT) à l'UFR de Mathématiques, Informatique, Mécanique de l'Université de Provence, Marseille.

Moniteur de l'enseignement supérieur.

Cursus universitaire

2001 **Doctorat en Informatique** de l'Université de Provence, encadré par Bélaïd BENHAMOU « Résolution du problème SAT et génération de modèles finis pour des logiques du premier ordre », Mention Très Honorable.

1998 **DEA d'Informatique** de l'Université de Provence, laboratoire d'accueil LIM, Mention Bien.

1996 **Maîtrise d'Informatique** de l'Université de Provence, Mention Bien.

1995 **Licence d'Informatique** de l'Université de Provence, Mention Très Bien.

1993 **Maîtrise de Mathématiques** de l'Université de Provence.

1992 **Licence de Mathématiques** de l'Université d'Aix Marseille 3, Mention Assez Bien.

Sommaire

Activités liées à la recherche

Thèmes de recherche

Le socle commun de mes activités de recherche est le problème SAT (Satisfaisabilité). Le caractère NP-complet de celui-ci en fait un problème intraitable dans le cas général. Malgré tout, de nombreux progrès théoriques et pratiques ont été réalisés ces dernières années. Ils ont fait passer l'utilisation de SAT pour montrer que des problèmes sont NP-complets et donc difficiles, à l'utilisation de SAT pour résoudre des problèmes difficiles comme ceux issus de la planification, de la cryptographie...

Je m'intéresse également à des extensions du problème SAT. La première d'entre elles est le problème QBF (Quantified Boolean Formula). Les variables d'un problème QBF peuvent être quantifiées existentiellement ($\exists$), comme c'est le cas dans un problème SAT, mais aussi universellement ($\forall$). Cela permet une représentation plus concise de certains problèmes aux dépens de difficultés supplémentaires pour les démonstrateurs.

Une autre extension du problème SAT à laquelle je porte de l'intérêt est le problème SMT (SAT modulo Théories). Un problème SMT contient des atomes propositionnels classiques mais également des atomes provenant d'autres théories. Les clauses obtenues peuvent, par exemple, être de la forme $\neg x \vee (v_1 = v_2 - 4) \vee lire(s, b - c) = d)$. Elles offrent donc une expressivité plus importante, mais, ici aussi, au détriment de difficultés supplémentaires dans la résolution.

Mes activités de recherche sont principalement orientées vers l'amélioration pratique de la résolution de ces trois problèmes. Cela passe par la proposition de nouvelles techniques, de nouveaux démonstrateurs et de nouvelles formes de représentation.

Activités d'évaluation de la recherche

- Membre de la **commission de spécialistes** (sections 25, 26, 27) de l'université d'Artois de 2003 à 2006, réélu pour la période 2007–2008.
- **Membre de comité de programmes**
  - 10èmes Journées nationales sur la résolution des problèmes NP-Complets (JNPC 2004)
  - 7èmes Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA 2005)
  - Journées Francophones de la Programmation par Contraintes : JFPC 2005, JFPC 2006 et JFPC 2007
  - workshop SymCom 2007 (Symmetry in Constraints, affilié à CP 2007) et RCRA 2008 et 2009
- **Relecteur** pour
  - Les journaux JSAT, RAIRO, Constraints, Computer Journal, Discrete Applied Mathematics
  - Les conférences internationales CALCULEMUS 2003, SAT 2003, SAT 2004, SAT 2007, IJCAI 2007 ECAI 2008
  - La conférence nationale JFPC 2010
- **Examinateur** des thèses de Mohamed REDA SAIDI (Université de Provence) et Saïd JABBOUR (Université d'Artois).

Participation à des projets de recherche

2009 – ...	ANR blanc UNLOC « Local Search and Unsatisfiability ». **Porteur local** du projet.
2005 – 2008	ANR jeunes chercheurs PLANEVO « Planification en milieu incertain et évolutif ».
2002 – 2004	Action Spécifique QBF, en partenariat avec les laboratoires de Toulouse (IRIT), Paris (LRI), Amiens (LARIA) et le CRIL
2002 – 2003	Action Universitaire Intégrée Luso–Françaises, projet « OpenSAT : a platform open source for SAT », en partenariat avec le groupe SAT du laboratoire INESC, Lisbonne Portugal.

Co-encadrement de thèses

2008 – ... Jean-Marie LAGNIEZ « Recherche Locale pour SAT et UNSAT ». Co-encadrant (33 %) avec B. MAZURE (33 %) sous la direction de Lakhdar SAÏS (33 %).
Articles en collaboration : [ALMS09b, ALMS09a, ALMS09c, ALM10, ALMS10]

2005 – 2008 Saïd JABBOUR « De la Satisfiabilité Propositionnelle aux Formules Booléennes Quantifiées ». Co-encadrant (50 %) sous la direction de Lakhdar SAÏS (50 %). Saïd JABBOUR est aujourd'hui Maître de Conférences à l'Université d'Artois.
Articles en collaboration : [AJS06, AJS07c, AJS07b, AJS07a, ABH+08a, ABH+08b, AJS08]

Encadrement de stages de master recherche ou DEA

2008 Jean-Marie LAGNIEZ « Recherche locale pour SAT et UNSAT ».

2005 Saïd JABBOUR « Simplification de QBF ».

2005 Gabriel WESTRELIN « Symétries dans les noyaux des Formules 3-SAT aléatoires »

2004 Jérome DEGAVE « Symétries et QBF ».

Distinction scientifique

2009 Compétition SAT'09 : démonstrateur GLUCOSE classé premier dans la catégorie application, instances UNSAT et second dans la catégorie application instances SAT+UNSAT.

Autres activités liées à la recherche

2005 **Membre du bureau** de l'Association Française pour la Programmation par Contraintes AFPC

2005 **Membre du comité d'organisation** des Premières Journées Francophones de la Programmation par Contraintes (JFPC).

Sommaire

Activités liées à l'enseignement

Synthèse des enseignements

En poste depuis 2002 au sein du département informatique de l'IUT de Lens, j'ai enseigné à des étudiants de DUT et de licence professionnelle option sécurité informatique. Depuis mon arrivée, j'ai enseigné environ 220 heures d'équivalent TD par an. Je ne vais pas lister toutes les matières où je suis intervenu, mais juste mettre en avant celles dont je suis (ou ai été) responsable depuis 2002.

❑	2005 – . . .	Programmation Web – partie 1 HTML – CSS – PHP	DUT1
❑	2006 – . . .	Programmation Web – partie 2 XML – flux RSS – javascript – Ajax	DUT2
❑	2008 – . . .	Modélisation Objet Diagrammes de classes, d'activité – Design patterns	DUT2
❑	2006 – . . .	Sécurité WEB configuration apache – faille WEB : XSS, CSRF, injections SQL	Licence Pro
❑	2004 – . . .	Sécurité des Programmmes Exceptions – test unitaires – buffer overflow. . .	Licence Pro
❑	2002 – 2007	Programmation Orientée Objet encapsulation – surcharge – héritage – polymorphisme	DUT2
❑	2002 – 2006	Structures de données avancées pile – file – liste – arbres	Année spéciale
❑	2002 – 2004	Bases de données – partie 1 requêtes SQL	DUT1
❑	2002 – 2003	Bases de données – partie 2 création tables – insertion données – triggers	DUT2

Projets tutorés

Les étudiants de DUT et de Licence professionnelle doivent réaliser des projets tutorés durant leur scolarité. Ces projets, réalisés par groupe de 3/4 personnes, sont proposés par des enseignants qui se chargent ensuite de les encadrer. Voici une liste exhaustive des divers projets que j'ai proposés et encadrés ces dernières années :

- Étudiants de DUT
  - Clone du jeu Warblade (Shoot'Em'Up).
  - Compression et décompression d'images par ondelettes.
  - Gestion des absences des étudiants du département (MySQL, PHP...). Je maintiens toujours cette plateforme et elle est utilisée dans 3 départements de l'IUT de Lens.
  - Site web de gestion de morceaux musicaux.
  - Site web de gestion de bookmarks.

- Étudiants de Licence professionnelle
  - Proxy parental devant filtrer des adresses, des pages contenant des mots interdits. . .
  - Réalisation d'une distribution LINUX sécurisée.
  - Création d'un laboratoire de langue.
  - Framework MVC sécurisé pour PHP.
  - Tests unitaires et PHP.

Animations

- Organisation d'une concours de programmation basé sur les robocodes (`http://robocode.sourceforge.net`) pour le dixième anniversaire de l'IUT de Lens (2003)
- Chaque année, organisation et animation de l'install party (ubuntu) pour les étudiants arrivant à l'IUT.
- Séminaire de vulgarisation de la cryptographie dans un lycée de Roubaix en 2010

Responsabilités administratives liées à l'enseignement

2009 – . . .	directeur des études des étudiants de deuxième année d'informatique de l'IUT de Lens
2003 – 2007	directeur des études des étudiants d'*Année Spéciale informatique* de l'IUT de Lens.

Sommaire

Liste des publications

Synthèse

- 1 chapitre de livre : « Problème SAT : progrès et défis »
- 3 articles parus dans des revues internationales
- 20 communications dans des conférences internationales avec comité de lecture (SAT, IJCAI, CADE...)
- 6 communications à des workshops internationaux
- 12 communications à des conférences nationales (JNPC, JFPC. . .)

Chapitre de Livre

[1] Gilles AUDEMARD and Belaid BENHAMOU.
« Problème SAT : progrès et défis ».
Chapitre Symétries. Hermes Sciences Publishing Ltd, 2008.

Revues internationales avec comité de rédaction

[1] Gilles AUDEMARD, Said JABBOUR, and Lakhdar SAÏS.
« SAT graph based representation : A new perspective ».
Journal of Algorithms in Cognition, Informatic and Logic, 63 :17–33, 2008.

[2] Gilles AUDEMARD, Belaid BENHAMOU, and Laurent HENOCQUE.
« Predicting and detecting symmetries in FOL finite model search ».
Journal of Automated Reasoning, 36(3) :177–212, 2006.

[3] Gilles AUDEMARD, Marco BOZZANO, Alessandro CIMATTI, and Roberto SEBASTIANI. « Verifying industrial hybrid systems with MathSAT ». *Electronic Notes in Theorical Computuer Science*, 119(2) :17–32, 2005.

Conférences internationales avec comité de lecture et actes

[1] Gilles AUDEMARD, George KATSIRELOS, and Laurent SIMON.
« A restriction of extended resolution for clause learning sat solvers ».
In *24nd Conference on Artificial Intelligence(AAAI'10)*, pages 15–20, 2010.

[2] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS.
« Boosting local search thanks to CDCL ».
In *17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning(LPAR'10)* , pages 474–488, 2010.

[3] Gilles AUDEMARD and Laurent SIMON.
« Predicting learnt clauses quality in modern SAT solvers ».
In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009.

[4] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS.
« Learning in local search ».
In *21st International Conference on Tools with Artificial Intelligence(ICTAI'09)*, pages 417–424, 2009.

[5] Gilles AUDEMARD and Laurent SIMON.
« Experimenting with small changes in conflict-driven clause learning algorithms ».
In *proceedings of Principles and Practice of Constraint Programming, 14th International Conference (CP)*, volume 5202 of *LNCS*, pages 630–634, 2008.

[6] Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, and Lakdhar SAÏS.
« A generalized framework for conflict analysis ».
In *proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4996 of *LNCS*, pages 21–27, 2008.

[7] Gilles AUDEMARD and Lakhdar SAÏS.
« Circuit based encoding of CNF formula ».
In *proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4501 of *LNCS*, pages 16–21, 2007.

[8] Gilles AUDEMARD, Saïd JABBOUR, and Lakhdar SAÏS.
« Symmetry breaking in quantified boolean formulae ».
In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2262–2267, 2007.

[9] Gilles AUDEMARD and Laurent SIMON.
« Gunsat : A greedy local search algorithm for unsatisfiability ».
In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2256–2261, 2007.

[10] Gilles AUDEMARD and Lakdhar SAÏS.
« A symbolic search based approach for quantified boolean formulas ».
In F. Bacchus T. Walsh, editor, *proceedings of the eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 16–30, 2005.

[11] Gilles AUDEMARD and Lakdhar SAÏS.
« SAT based BDD solver for quantified boolean formulas ».
In *proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 82–89, 2004.

[12] Gilles AUDEMARD, Bertrand MAZURE, and Lakdar SAÏS.
« Dealing with symmetries in quantified boolean formulas ».
In *proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 257–262, 2004.

[13] Gilles AUDEMARD, Daniel LE BERRE, Olivier ROUSSEL, Ines LYNCE, and Joao MARQUES SILVA.
« Opensat : An open source SAT open project ».
In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2003.

[14] Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI.
« A sat based approach for solving formulas over boolean and linear mathematical propositions ».
In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, pages 195–210, 2002.

[15] Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI.
« Integrating boolean and mathematical solving : Foundations, basic algorithms, and requirements ».
In *Proceedings of the Joint International Conference, AISC 2002 and Calculemus 2002*, volume 2385 of *LNCS*, pages 231–245. Springer Verlag, 2002.

[16] Gilles AUDEMARD, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI.
« Bounded model checking for timed systems ».
In *Proceedings of the International Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, pages 243–259. Springer Verlag, 2002.

[17] Gilles AUDEMARD and Belaid BENHAMOU.
« Reasoning by symmetry and function ordering in finite model generation ».
In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, pages 226–240. Springer Verlag, 2002.

[18] Gilles AUDEMARD and Laurent HENOCQUE.
« The eXtended Least Number Heuristic ».
In T. Nipkow R. Goré, A. Leitsch, editor, *Proceedings of the first International Join Conference in Automated Reasonning (IJCAR)*, volume 2083 of *LNCS*, pages 427–442. Springer Verlag, 2001.

[19] Gilles AUDEMARD, Belaid BENHAMOU, and Laurent HENOCQUE.
« Two techniques to improve finite model search ».
In David McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, volume 1831 of *LNCS*, pages 302–308. Springer Verlag, 2000.

[20] Gilles AUDEMARD, Belaid BENHAMOU, and Pierre SIEGEL.
« AVAL : An enumerative method for SAT ».
In *Proceedings of first international conference on computational logic CL00, Londres*, volume 1861 of *LNCS*, pages 373–383. Springer Verlag, 2000.

Conférences internationales avec comité de lecture sans actes

[1] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS.
« Integrating conflict driven clause learning to local search ».
In *International Workshop on Local Search Techniques in Constraint Satisfaction (affiliated to CP)(LSCS09)*, 2009.

[2] Gilles AUDEMARD, Said JABBOUR, and Lakdhar SAÏS.
« Using SAT-graph representation to derive hard instances ».
In *The 14th RCRA workshop Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2007.

[3] Gilles AUDEMARD, Said JABBOUR, and Lakdhar SAÏS.
« Efficient symmetry breaking predicates for quantified boolean formulae ».
In *7th International Workshop on Symmetry and Constraint Satisfaction Problems - Affiliated to CP*, 2007.

[4] Gilles AUDEMARD, Marco BOZZANO, Alessandro CIMATTI, and Roberto SEBASTIANI.
« Verifying industrial hybrid systems with MathSAT ».
In *Workshop BMC - Affiliated to CAV*, 2004.

[5] Gilles AUDEMARD, Marco BOZZANO, Alessandro CIMATTI, and Roberto SEBASTIANI.
« Verifying industrial hybrid systems with MathSAT ».
In *Proceedings of the first workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR)*, pages 62 – 75, 2003.

[6] Gilles AUDEMARD and Belaid BENHAMOU.
« Symmetry in finite model of first order logic ».
In *SymCon'01 – Symmetry in Constraints - CP01 Workshop*, 2001.

Conférences nationales avec comité de lecture et actes

[1] Gilles AUDEMARD , Jean-Marie LAGNIEZ, and Bertrand MAZURE.
« Approche hybride pour SAT ».
In *17eme congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle(RFIA'10)*, pages 279–286, 2010.

[2] Gilles AUDEMARD , George KATSIRELOS, and Laurent SIMON.
« Une restriction de la résolution étendue pour les démonstrateurs SAT modernes ».
In *Sixièmes Journées Francophones de Programmation par Contraintes(JFPC'10)*, pages 43–50, 2010.

[3] Gilles AUDEMARD , Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS.
« Analyse de conflits dans le cadre de la recherche locale ».
In *Journées Francophones de la Programmation par Contraintes(JFPC'09)*, pages 215–224, 2009.

[4] Gilles AUDEMARD , Mouny Samy-Modeliar, and Laurent SIMON.
« Pourquoi les solveurs sat modernes se piquent-ils contre des cactus ? ».
In *Journées Francophones de la Programmation par Contraintes(JFPC'09)*, pages 245–253, 2009.

[5] Gilles AUDEMARD , Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, and Lakhdar SAÏS.
« Un cadre général pour l'analyse des conflits ».
In *Actes des quatrièmes Journées Francophones de Programmation par Contraintes(JFPC'08)*, pages 267–276, 2008.

[6] Gilles AUDEMARD , Said JABBOUR, and Lakdhar SAÏS.
« Exploitation des symétries dans les formules booléennes quantifiées ».
In *journées francophones de la programmation par contraintes (JFPC)*, 2006.

[7] Gilles AUDEMARD and Lakdhar SAÏS.
« Une approche symbolique pour les formules booléennes quantifiées ».
In *journées francophones de la programmation par contraintes (JFPC)*, pages 59–68, 2005.

[8] Gilles AUDEMARD , Bertrand MAZURE, and Lakdhar SAÏS.
« Symétries et formules booléennes quantifiées ».
In *journées nationales des problèmes NP-complets (JNPC)*, 2004.

[9] Gilles AUDEMARD , Daniel LE BERRE, and Olivier ROUSSEL.
« OpenSAT : Une plateforme SAT open source ».
In *Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC) - Amiens*, 2003.

[10] Gilles AUDEMARD and Belaid BENHAMOU.
« Étude des symétries dans les modèles finis ».
In *Journées Francophones de la Programmation en Logique et Programmation par Contraintes (JFPLC'2001)*, pages 109–122. Hermès Sciences, 24–27 2001.

[11] Gilles AUDEMARD and Laurent HENOCQUE.
« Sur la génération des groupes finis non isomorphes ».
In *Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC) - Marseillle*, pages 57–66, 2000.

[12] Gilles AUDEMARD , Belaid BENHAMOU, and Pierre SIEGEL.
« La méthode d'avalanche AVAL : une méthode énumérative pour SAT ».
In *Journées Nationales de la Résolution Pratique des problèmes NP-Complets (JNPC)*, pages 17–25, 1999.

Mémoire de thèse et DEA

[1] Gilles AUDEMARD
« Résolution du problème SAT et génération de modèles finis pour des logiques du premier ordre »
Thèse de doctorat, Université de Provence, Marseille, octobre 2001.

[2] Gilles AUDEMARD
« Avalanche dans 3-SAT »
Mémoire de DEA, Université de Provence, Marseille, juin 1998.

Partie III

Sélection de publications

Liste des Articles

Autour de SAT

- **Solveurs CDCL**
  Gilles AUDEMARD and Laurent SIMON. « Predicting Learnt Clauses Quality in Modern SAT solvers ». In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009.

  Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, and Lakdhar SAÏS. « A generalized framework for conflict analysis ». In *proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4996 of *LNCS*, pages 21–27, 2008.

- **Recherche locale est insatisfaisabilité**
  Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS. « Learning in local search ». In *21st International Conference on Tools with Artificial Intelligence(ICTAI'09)*, pages 417–424, 2009.

- **Travaux connexes**
  Gilles AUDEMARD, Said JABBOUR, and Lakhdar SAÏS. « SAT graph based representation : A new perspective ». *Journal of Algorithms in Cognition, Informatic and Logic*, 63 :17–33, 2008.

Autour de QBF

- **Suppression des symétries**
  Gilles AUDEMARD, Saïd JABBOUR, and Lakhdar SAÏS. « Symmetry breaking in quantified boolean formulae ». In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2262–2267, 2007.

- **Un solveur basé sur les BDDs**
  Gilles AUDEMARD and Lakdhar SAÏS. « A symbolic search based approach for quantified boolean formulas ». In *proceedings of the eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 16–30, 2005.

Autour de SMT

– **La méthode MATHSAT**
Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI. « A sat based approach for solving formulas over boolean and linear mathematical propositions ». In *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, pages 195–210, 2002.

– **Application à la vérification d'automates temporels**
Gilles AUDEMARD, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI. « Bounded model checking for timed systems ». In *Proceedings of the International Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, pages 243–259. Springer Verlag, 2002.

Predicting Learnt Clauses Quality in Modern SAT solvers

Gilles AUDEMARD and Laurent SIMON.

In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009.

Predicting Learnt Clauses Quality in Modern SAT Solvers*

Gilles Audemard
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307
audemard@cril.fr

Laurent Simon
Univ. Paris-Sud
LRI/CNRS UMR 8623 / INRIA Saclay
Orsay, F-91405
simon@lri.fr

Abstract

Beside impressive progresses made by SAT solvers over the last ten years, only few works tried to understand why Conflict Directed Clause Learning algorithms (CDCL) are so strong and efficient on most industrial applications. We report in this work a key observation of CDCL solvers behavior on this family of benchmarks and explain it by an unsuspected side effect of their particular Clause Learning scheme. This new paradigm allows us to solve an important, still open, question: How to designing a fast, static, accurate, and predictive measure of new learnt clauses pertinence. Our paper is followed by empirical evidences that show how our new learning scheme improves state-of-the art results by an order of magnitude on both SAT and UNSAT industrial problems.

1 Introduction

Only fifteen years ago, the SAT problem was mainly considered as theoretical, and polynomial reduction to it was a tool to show the intractability of any new problem. Nowadays, with the introduction of lazy data-structures, efficient learning mechanisms and activity-based heuristics [Moskewicz *et al.*, 2001; Eén and Sörensson, 2003], the picture is quite different. In many combinatorial fields, the state-of-the-art schema is often to use an adequate polynomial reduction to this canonical NP-Complete problem, and then to solve it efficiently with a SAT solver. This is especially true with the help of Conflict Directed Clause Learning algorithms (CDCL), an efficient SAT algorithmic framework, where our work takes place. With the help of those algorithms, valuable problems [Prasad *et al.*, 2005] are efficiently solved every day.

However, the global picture is not so perfect. For instance, since the introduction of ZCHAFF [Moskewicz *et al.*, 2001] and MINISAT [Eén and Biere, 2005], the global architecture of solvers is stalling, despite important efforts of the whole community [Le Berre *et al.*, 2007]. Of course, improvements have been made, but they are sometimes reduced to data-structures tricks. It may be thus argued that most solvers are often described as a compact re-encoding of ideas of Chaff, a height-years old solver, with small improvements only (phase caching [Pipatsrisawat and Darwiche, 2007], luby restarts [Huang, 2007], ...).

Since the breakthrough of Chaff, most effort in the design of efficient SAT solvers has always been focused on efficient Boolean Constraint Propagation (BCP), the heart of all modern SAT solvers. The global idea is to reach conflicts as soon as possible, but with no guarantees on the new learnt clause usefulness. Following the successful idea of the Variable State Independent Decaying Sum (VSIDS) heuristics, which favours variables that were often – and recently – used in conflict analysis, future learnt clause usefulness was supposed to be related to its activity in recent conflicts analyses.

In this context, detecting what is a good learnt clause in advance is still considered as a challenge, and from first importance: deleting useful clauses can be dramatic in practice. To prevent this, solvers have to let the maximum number of learnt clauses grow exponentially. On very hard benchmarks, CDCL solvers hangs-up for memory problems and, even if they don't, their greedy learning scheme deteriorates their heart: BCP performances.

We report, in section 2, experimental evidences showing an unsuspected phenomenon of CDCL solvers on industrial problems. Based on these observations, we present, in section 3, our static measure of learnt clause usefulness and, in section 4, we discuss the introduction of our measure in CDCL solvers. In section 5, we compare our solver with state-of-the-art ones, showing order of magnitude improvements.

2 Decision levels regularly decrease

Writing an experimental study in the first section of a paper is not usual. Paradoxically, CDCL solvers lack for strong empirical studies: most papers contain experimental sections, but focus is often given to achievement illustrations only.

In this section, we emphasize an observation – decision level is decreasing – made on most CDCL solvers on most industrial benchmarks. This behavior may shed a new light on the underlying reasons of the good performances of the CDCL paradigm. The relationship between performances and decreasing is the basis of our work in the following sections.

For lack of space, we suppose the reader familiar with Satisfiability notions (variables x_i, literal x_i or $\neg x_i$, clause, unit clause and so on). We just recall the global schema of CDCL

*supported by ANR UNLOC project n° BLAN08-1_328904

Series	#Benchs	% Decr.	$-n/m(>0)$	Reduc.
een	8	62%	1.1×10^3	1762%
goldb	11	100%	1.4×10^6	93%
grieu	7	71%	1.3×10^6	–
hoons	5	100%	7.2×10^4	123%
ibm-2002	7	71%	4.6×10^4	28%
ibm-2004	13	92%	1.9×10^5	52%
manol-pipe	55	91%	1.9×10^5	64%
miz	13	0%	–	–
schup	5	80%	4.8×10^5	32%
simon	10	90%	1.1×10^6	50%
vange	3	66%	4.0×10^5	6%
velev	54	92%	1.5×10^5	81%
all	199	83%	3.2×10^5	68%

Table 1: $x_j = -n/m$ (fourth column) is the positive look-back "justification" of the number of conflicts needed to solve benchmarks, median value over all benchmarks that show a decreasing of their decision level (%Decr). Last column will be discussed in section 5.

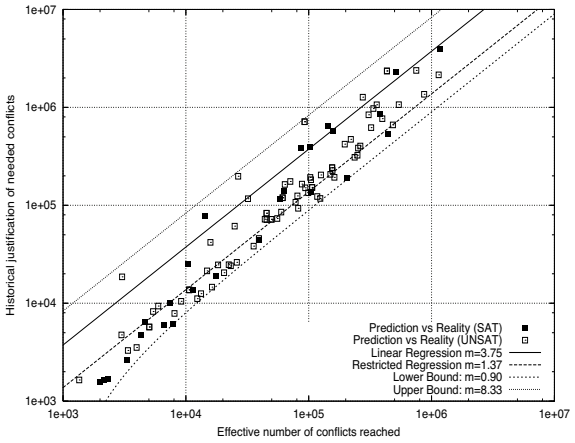


Figure 1: Relationship between look-back justification and effective number of conflicts needed by MINISAT to solve the instance.

solvers: A typical branch of a CDCL solver can be seen as a sequence of decisions followed by propagations, repeated until a conflict is reached. Each decision literal is assigned at its own level (starting from 1), shared with all propagated literals assigned at the same level. Each time a conflict is reached, a *nogood* is extracted using a particular method, usually the First UIP (Unique Implication Point) one [Zhang and Madigan, 2001]. The learnt clause is then added to the clause database and a *backjumping* level is computed from it. The interested reader can refer to [Marques-Silva *et al.*, 2009] for more details.

2.1 Observing the decreasing

The experimental study is done as follows. We run MINISAT on a selection of benchmarks from last SAT contests and races. For a given benchmark, each time a conflict x_c is reached, we store the decision level y_l where it occurs. We limit the search to 2 million of conflicts. Then, we compute the simple least-square linear regression on the line $y = m \times x + n$ that fits the set of couples (x_c, y_l). If m is negative (resp. positive) then decision levels decrease during search (resp. increase). In case of decreasing, we can trivially "*predict*" when the solver will finish the search. This should occur, if the solver follows its characteristic line, when this line intersects the x-axis. We call this point the "*look-back justification*" of the solver performance (looking-back is necessary to compute the characteristic line). The coordinates of this point are $(x_j = -n/m, 0)$. This value gives also a good intuition of how decision levels decrease during search. Note that we make very strong hypotheses here: (1) the solver follows a linear decreasing of its decision levels (this is false in most of the cases, but sufficient to compute the look-back justification), (2) finding a contradiction or a solution gives the same look-back justification and (3) the solution (or contradiction) is not found by chance at any point of the computation.

Table 1 shows the median values of x_j (fourth column) over some series of benchmarks. "#Benchs" gives the number of benchmarks in series, "%Decr." gives the percentage of benchmarks that exhibits a decreasing of decision levels. If a cut-off occurs, then the last value of the estimation is taken. In most of the cases (167 over 199 benchmarks), the regression line is decreasing, and some small values of x_0 illustrates important ones. This phenomenon was not observed on random problems and seems heavily related to the "industrial" origin of benchmarks: the "mizh" series is 100% increasing, but it encodes cryptographic problems.

2.2 Justifying the number of reached conflicts

At this point of the discussion, we showed that decision levels are decreasing in most of the cases. However, is it really a strong – but unsuspected – explanation of the power of CDCL solvers, or just a natural side effect of their learning schema? It is indeed possible that learning clauses trivially add more constraints, and conflicts are obviously reached at lower and lower levels in the search tree. However, if the look-back justification is a strong estimation of the effective number of conflicts needed by the solver to find the contradiction (or, amazingly, a solution), then this probably means that the decreasing reveals an important characteristic of their overall behavior and strength: CDCL solvers enforce the decision level to decrease along the computation, whatever the fitting quality of the linear regression.

Figure 1 exhibits the relationship between look-back justification and effective number of conflicts needed by MINISAT to solve the instance. It is built as follows: Each dot (x, y) represents an instance. x corresponds to the effective number of conflicts needed by MINISAT to solve the instance, y corresponds to the look-back justification. Only instances solved in less than 2 million of conflicts are represented (otherwise, we do not have the real look-back justification). Figure 1 clearly shows the strong relationship between justification and effective number of conflicts needed: in all the cases, the look-back justification is bounded between 0.90 and 8.33 times the real number of conflicts needed by MINISAT to solve the problem. In most of the cases, the justification is

around 1.37 times the effective number of conflicts. On may also notice that no distinction can be made between justification over SAT and UNSAT instances.

It is important here to notice that the aim of our study is not to predict the CPU time of the solver like in [Hutter *et al.*, 2006]. Our goal is, whatever the error obtained when computing the regression line, to show the strong relationship between the overall decreasing of decision levels and the performances of the solver. What is really striking is that the look-back justification is also strong when the solver finds a solution. When a solution exists, it is never found suddenly, by chance. This suggests that, on SAT instances, the solver does not correctly guess a value for a literal, but learns that the opposite value directly leads to a contradiction. Thus, if we find the part of the learning schema that enforces this decreasing, we may be able (1) to speed-up the decreasing, and thus the CPU time of the solver, and (2) to identify in advance the clauses that play this particular role for protection and aggressive clause database deletion.

3 Identifying good clauses in advance

Trying to enhance the decreasing of decision levels during search is not new. It was already pointed out, but from a general perspective only, in earlier papers on CDCL solvers: " *A good learning scheme should reduce the number of decisions needed to solve certain problems as much as possible*." [Zhang and Madigan, 2001]. In the previous section, it was however demonstrated for the first time how crucial this ability is for CDCL solvers.

3.1 Some known results on CDCL behavior

Let us first recall some high level principles of CDCL solvers, by firstly focusing on their restart policies. In earlier works, restarting was proposed as an efficient way to prevent heavy-tailed phenomena [Gomes *et al.*, 2000]. However, in the last years, restart policies were more and more aggressive, and, now, a typical run of a state-of-the art CDCL solver sees most of its restarts occurring before the 1000th conflict [Huang, 2007; Biere, 2008] (this strategy especially pays when it is associated to phase savings [Pipatsrisawat and Darwiche, 2007]). So, in a few years, the concept of restarting has saw his meaning moving from "restart elsewhere" (trying to reach an easier contradiction elsewhere in the search space) to "restart dependencies order" that just reorder variable dependencies, leading the solver to the same search space, by a different path ([Biere, 2008] has for instance already pointed out this phenomenon).

In addition to the above remark, it is striking to notice how CDCL solvers are particularly efficient on the so-called "industrial" benchmarks. Those benchmarks share a very particular structure: they often contain very small backdoors sets [Williams *et al.*, 2003], which, intuitively, encode inputs of formulas. Most of the remaining variables directly depend on their values. Those dependencies implicitly links sets of variables that depend on the same inputs. During search, those "linked" variables will probably be propagated together again and again, especially with the locality of the search mechanism of CDCL solvers.

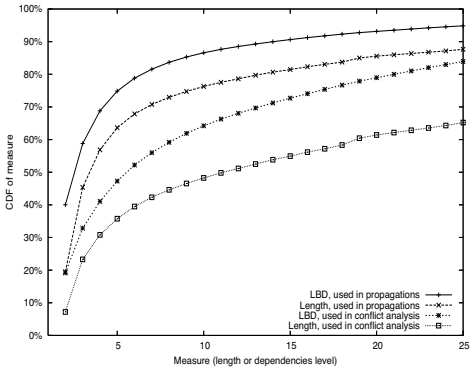


Figure 2: CDF of usefulness of clauses w.r.t. LBD and size.

3.2 Measuring learnt clause quality

During search, each decision is generally followed by a number of unit propagations. All literals from the same level are what we call "blocks" of literals in the later. At the semantic level, there is a chance that they are linked with each other by direct dependencies. Thus, a good learning schema should add explicit links between independent blocks of propagated (or decision) literals. If the solver stays in the same search space, such a clause will probably help reducing the number of next decision levels in the remaining computation.

Definition 1 (Literals Blocks Distance (LBD)) *Given a clause C, and a partition of its literals into n subsets according to the current assignment, s.t. literals are partitioned w.r.t their decision level. The LBD of C is exactly n.*

From a practical point of view, we compute and store the LBD score of each learnt clause when it is produced. This measure is static, even if it is possible (we will see in the later) to update it during search. Intuitively, it is easy to understand the importance of learnt clauses of LBD 2: they only contain one variable of the last decision level (they are FUIP), and, later, this variable will be "glued" with the block of literals propagated above, no matter the size of the clause. We suspect all those clauses to be very important during search, and we give them a special name: "Glue Clauses".

3.3 First results on the literals blocks distance

We first ran MINISAT on the set of all SAT-Race 06[1] benchmarks, with a CPU cut off fixed at 1000s. For each learnt clause, we measured the number of times it was useful in unit-propagation and in conflict analysis. Figure 2 shows the cumulative distribution function of the values over all benchmarks (statistics over unfinished jobs were also gathered). For instance, 40% of the unit propagations on learnt clauses are done on glue clauses, and only 20% are done on clauses of size 2. Half of the learnt clauses used in the resolution mechanism during all conflict analysis have LBD < 6, whereas we need to consider clauses of size smaller than 13 for the same result. Let's look at some details on table 2. What is striking is how the usefulness drops from 1827 times in conflict

[1] Used as a basis for the SAT-Race 08 too.

Measure	2	3	4	5
LBD	202 / 1827	50 / 295	26 / 136	19 / 80
Length	209 / 2452	127 / 884	46 / 305	33 / 195

Table 2: Average number of times a clauses of a given measure was used (propagation/conflict analysis).

analysis for glue clauses to 136 for clauses of LBD 4. We ran some additional experiments to ensure that clauses may have a large size (hundreds of literals) and a very small LBD.

From a theoretical point of view, it is interesting to notice that LBD of FUIP learnt clause is optimal over all other possible UIP learning schemas [Jabbour and Sais, 2008]. If our empirical study of this measure shows its accuracy, this theoretical result will cast a good explanation of the efficiency of First UIP over all other UIP mechanisms (see for instance [Marques-Silva *et al.*, 2009] for a complete survey of UIP mechanisms): FUIP efficiency would then be partly explained by its ability to produce clauses of small LBD.

Property 1 (Optimality of LBD for FUIP Clauses) *Given a conflict graph, any First UIP asserting clause has the smallest LBD value over all other UIPs.*

sketch of proof: This property was proved as an extension of [Audemard *et al.*, 2008]. When analyzing the conflict, no decision level can be deleted by resolution on a variable of the same decision level, because the resolution is done with *reasons* clauses: if a variable is propagated at level l, then the reason clause contains also another variable set at the same level l, otherwise the variable would have been set at a lower decision level $< l$, by unit propagation.

4 Agressive clauses deletion

CDCL solvers performances are tightly related to their clauses database management. Keeping too many clauses will decrease the BCP efficiency, but cleaning out to many ones will break the overall learning benefit. Most effort in the design of modern solvers is put in efficient BCP, leading to a very fast learnt clauses production. Following the success of the VSIDS heuristics, good learnt clauses are identified by their recent usefulness during conflicts analysis, despite the fact that this measure is not a guarantee of its future significance. This is why solvers often let the clauses set grow exponentially. This is a necessary evil in order to prevent good clause to be deleted. On hard instances, this greedy learning scheme deteriorates the core of CDCL solvers: their BCP performance drops down, making some problems yet much harder to solve.

Our aim in this section is to build an aggressive cleaning strategy, which will drastically reduce the learnt clauses database. This is possible especially if the LBD measure introduced above is accurate enough. No matter the size of the initial formula, we remove half of the learnt clauses (asserting clauses are kept) every $20000 + 500 \times x$ conflicts (x is the number of times this action was previously performed).

Table 3 summarizes the performances of four different versions of MINISAT: the original one, the original one with aggressive clause deletion (MINISAT +ag) based on the original clause activity, the classical one based on LBD (MINISAT +lbd) and the classical one with aggressive clause deletion based on LBD scoring (MINISAT +ag+lbd). This comparison is made on the set of 200 benchmarks from the SAT-Race 2006, with a time out of 1000s. We report the number of solved problems and the average time needed to solve them. It is firstly surprising to see that aggressive clause deletion can pay with the original MINISAT. However, less UNSAT benchmarks are solved, despite the overall speed-up dues to fast BCP rates. This may be explained by the deletion of too much *good* learnt clauses, that are essential for efficient UNSAT proofs. The version with LBD solves yet more instances, illustrating its efficiency. What is especially encouraging is the speedup observed between the original MINISAT and the one with our static LBD measure, despite the fact that three less UNSAT benchmarks are solved. Using our very simple and static measure in place of the dynamic one already pays a lot. Finally, the combination of both aggressive clause deletion and LBD is very strong.

	#N (sat-unsat)	#avg time
MINISAT	70 (35 – 35)	209
MINISAT +ag	74 (41 – 33)	194
MINISAT +lbd	79 (**47** – 32)	**145**
MINISAT +ag+lbd	**82** (45 – **37**)	175

Table 3: Comparison of four versions of MINISAT.

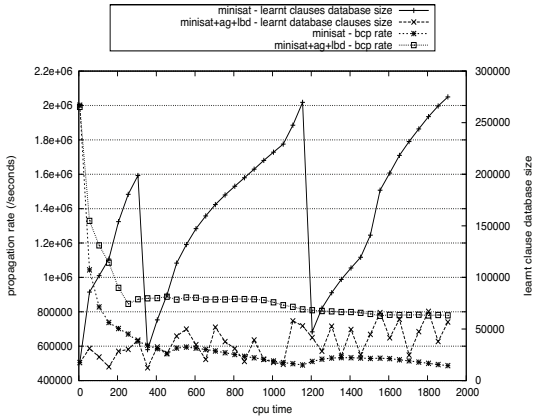


Figure 3: Comparison of BCP rate (left y scale) and learnt clauses database size (right y scale) between MINISAT and MINISAT +ag+lbd.

In order to justify these facts, figure 3 shows, for a selected hard UNSAT formula (with approximatively 50000 variables and 180000 clauses), the evolution of BCP rates and the size of the learnt clauses database. Values are gathered at regular clock time. This figure shows the huge explosion of learnt clauses in the original MINISAT. Only two calls to the reduction of the database are performed in 2000 seconds. In MINISAT +ag+lbd, an important number of database reductions are performed, keeping the BCP rate of MINISAT +ag+lbd much faster than MINISAT.

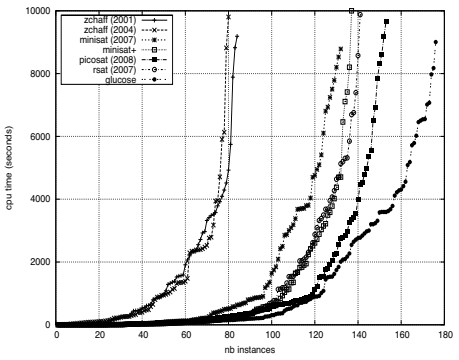


Figure 4: Cactus plot on the 234 industrial instances of the SAT'07 competition.

solver	#N	(SAT-UNSAT)	#U	#B	#S
ZCHAFF 01	84	(47 – 37)	0	13	2.9
ZCHAFF 04	80	(39 – 41)	0	5	3.9
MINISAT+	136	(66 – 74)	0	15	1.5
MINISAT	132	(53 – 79)	1	16	2.1
PICOSAT	153	(**75** – 78)	1	26	1.2
RSAT	139	(63 – 75)	1	14	1.7
GLUCOSE	**176**	(**75** – **101**)	**22**	**68**	-

Table 4: Relative performances of solvers. Column #N reports the number of solved benchmarks with, in parenthesis, the number of SAT and UNSAT instances. Column #U shows the number of times where the solver is the only one to solve an instance, column #B gives the number of times it is the fastest solver, and column #S gives the relative speed-up of GLUCOSE when considering only the subset of common solved instances between GLUCOSE and each solvers (for instance GLUCOSE is 1.7 times faster than RSAT on the subset of benchmarks they both solve).

5 Comparison with the best CDCL solvers

In this final section, we show how our ideas can be embedded in an efficient SAT solver. We used as a basis for it the well-known core of MINISAT using a Luby restarts strategy (starting at 32) with phase savings. We call this solver GLUCOSE for its hability to detect and keep "Glue Clauses". We added two tricks to it. Firstly, each time a learnt clause is used in unit-propagation (it is a reason of a propagated literal), we compute a new LBD score and update it if necessary. Secondly, we explicitly increase the score of variables of the learnt clause that were propagated by a glue clause.

We propose here to compare GLUCOSE with the three winners of the last SAT competition: MINISAT (version 2-070721) [Eén and Sörensson, 2003], RSAT (version 2.02) [Pipatsrisawat and Darwiche, 2007] and PICOSAT (version 846) [Biere, 2008]. We also add in this comparison ZCHAFF [Moskewicz *et al.*, 2001], the first CDCL solver (versions of years 2001 and 2004) and a version of MINISAT including luby restarts (starting at 100) and phase polarity (called MINISAT+ in the rest of the section, as the last – but unreleased – version of it, as it was described). We use the same (234) industrial benchmarks and running characteristics (10000s and 1Gb memory limit) than the SAT'07 competition [Le Berre *et al.*, 2007] in the second stage. According to well admitted idea, we pre-processed instances with SATELITE [Eén and Biere, 2005] and ran all solvers on these pre-processed instances. We used a farm of Xeon 3.2 Ghz with 2 Go RAM for this experiment.

Figure 4 shows the classical "*cactus*" plot used in SAT competitions. X-axis represents the number of instances and Y-axis represents the time needed to solve them if they were ran in parallel. First of all, comparing ZCHAFF and the last winners of the 2007 SAT competition, we can see the gap made during the last seven years. We can also remark that MINISAT, RSAT and MINISAT+ have approximatively the same performances. Before GLUCOSE, PICOSAT was the better solver. We see how GLUCOSE outperforms all of these solvers. It is able to solve 140 problems in 2500 seconds, when currently state-of-the-art solvers need between 4000 and 10000 seconds for this. One of the advantages of this kind of plots is its ability to emphasize when each solver reaches some kind of CPU time limit, when adding more CPU time doesn't add much chance to solve additional problems. For instance, RSAT solves approximatively 100 instances in 1000 seconds but only 40 more in 10000. It is not the case for GLUCOSE, which shows a better scaling up (it solves around 120 instances in 1000 seconds and 60 more in 10000). This is clearly due to our aggressive learnt clause database reduction combined with our clause usefulness measure (BCP performance is maintained and *good* clauses are kept).

Table 4 shows some detailed results. GLUCOSE solves 176 instances, for approximately 140 for RSAT, MINISAT and MINISAT+ (note that GLUCOSE needs only 2500 seconds to solve 140 instances) and 153 for PICOSAT. GLUCOSE especially shows impressive results on UNSAT instances, which really matches our initial motivation, i.e. enhancing the decision levels decreasing. However, we can assume that this strategy is also interesting on SAT instances, since GLUCOSE obtains, with PICOSAT, the best result on these instances. An interesting point is the number of times GLUCOSE is the only solver to solve a given problem (column #U). This shows the power of our method. Finally, the last two columns of table 4 show that GLUCOSE is much faster than all other solvers. It is the fastest solver in 68 cases and performs a speed-up on common instances of at least 1.2.

Before concluding this section, Figure 5 allows us to focus on the relative performances of GLUCOSE and PICOSAT which behaves very well (as shown table 4). The x-axis (resp. y-axis) corresponds to the cpu time tx (resp. ty) obtained by PICOSAT (resp. GLUCOSE). Each dot (tx, ty) matches the same benchmark. Thus, dots below (resp. above) the diagonal indicate that GLUCOSE is faster (resp. slower) than PICOSAT.

In many cases, GLUCOSE outperforms PICOSAT. When PICOSAT wins, GLUCOSE is still close to its performances. On UNSAT instances, and except a few cases, even when GLUCOSE looses, it is close to PICOSAT. Most bad performances are observed on SAT instances only. After a deeper

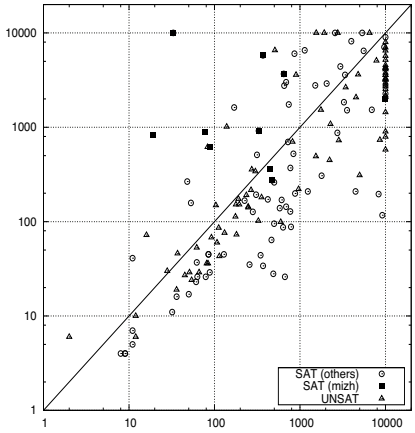


Figure 5: Comparison between GLUCOSE and PICOSAT.

look on those instances, we noticed that seven of those difficult instances are from the mizh family. This result is paradoxically yet again encouraging because, as pointed out in table 1, the evolution of decision levels of CDCL solvers on those instances is increasing, and thus does not match our initial aim: accelerating the decreasing of decision levels (let us recall here that mizh benchmarks are not "industrial" ones, in the classical sense).

In order to come full circle, let us explain now the last column of table 1. This percentage is the relative value of the look-back justification of GLUCOSE by comparison with MINISAT. For instance, on the ibm-2002 instances, the look-back justification of GLUCOSE is 28% smaller than the one of MINISAT. Thus, our overall strategy pays: we accelerated the decreasing of decision levels in most of the cases.

6 Conclusion

In this paper, we introduced a new static measure over learnt clauses quality that enhances the decreasing of decision levels along the computation on both SAT and UNSAT instances. This measure seems to be so accurate that very aggressive learnt clause database management is possible. We expect this new clause measure to have a number of great impacts. We may try to use it to propose preprocessing techniques (adding short LBD clauses), but also more exotic ones, like trying to reorder the formula in order to get a good starting of the decreasing. More classically, we think that our measure will also be very useful in the context of parallel SAT solvers. Indeed, such solver needs to share *good* clauses, which can be now identified with confidence by our measure.

Finally, we can now put efforts in designing an efficient framework for incomplete algorithm for Unsatisfiability. We know what efficient solvers should look for, and the future of our work is clearly on that path. We think that this work opens a new a promising path to answer one of the strongest challenges proposed more than ten years ago, in [Selman *et al.*, 1997], and still wide open.

References

[Audemard *et al.*, 2008] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A generalized framework for conflict analysis. In *proceedins of SAT*, pages 21–27, 2008.

[Biere, 2008] A. Biere. PicoSAT essentials. *Journal on Satisfiability*, 4:75–97, 2008.

[Eén and Biere, 2005] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.

[Gomes *et al.*, 2000] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[Huang, 2007] J. Huang. The effect of restarts on the efficiency of clause learning. In *proceedings of IJCAI*, pages 2318–2323, 2007.

[Hutter *et al.*, 2006] F. Hutter, Y. Hamadi, H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *proceedings of CP*, pages 213–228, 2006.

[Jabbour and Sais, 2008] S. Jabbour and L. Sais. personnal communication, February 2008.

[Le Berre *et al.*, 2007] D. Le Berre, O. Roussel, and L. Simon. SAT competition, 2007. http://www.satcompetition.org/.

[Marques-Silva *et al.*, 2009] J. Marques-Silva, I. Lynce, and S. Malik. *Handbook of Satisfiability*, chapter 4. 2009.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.

[Pipatsrisawat and Darwiche, 2007] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In *proceedings of SAT*, pages 294–299, 2007.

[Prasad *et al.*, 2005] M. Prasad, A. Biere, and A. Gupta. A survey of recent advances in SAT-based formal verification. *journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[Selman *et al.*, 1997] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.

[Williams *et al.*, 2003] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI'03*, 2003.

[Zhang and Madigan, 2001] L. Zhang and C. Madigan. Efficient conflict driven learning in a boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.

A generalized framework for conflict analysis

Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, and Lakdhar SAÏS.

In *proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 4996 of *LNCS*, pages 21–27, 2008.

A Generalized Framework for Conflict Analysis

G. Audemard[1], L. Bordeaux[2], Y. Hamadi[2], S. Jabbour[1], and L. Sais[1]

[1] CRIL - CNRS UMR 8188, Artois, France
{audemard,jabbour,sais}@cril.fr
[2] Microsoft Research Cambridge, UK
{lucasb,youssefh}@microsoft.com

Abstract. This paper presents an extension of Conflict Driven Clauses Learning (CDCL). It relies on an extended notion of implication graph containing additional arcs, called inverse arcs. These are obtained by taking into account the satisfied clauses of the formula, which are usually ignored by conflict analysis. This extension captures more conveniently the whole propagation process, and opens new perspectives for CDCL-based approaches. Among other benefits, our extension leads to a new conflict analysis scheme that exploits the additional arcs to back-jump to higher levels. Experimental results show that the integration of our generalized conflict analysis scheme within two state-of-the-art solvers improves their performance.

1 Introduction

This paper extends *Conflict-Driven Clause-Learning (CDCL)*, which is one of the key components of modern SAT solvers [7,5]. In the CDCL approach a central data-structure is the *implication graph*, which records the partial assignment that is under construction together with its implications. This data-structure enables conflict analysis, which, in turn, is used for intelligent backtracking, clause learning, for the adjustment of the variable selection heuristic. An important observation is that the implication graph built in the traditional way is "incomplete" in that it only gives a partial view of the actual implications between literals. A solver only keeps track of the first explanation that is encountered for the deduced literal. This strategy is obviously very much dependent on the particular order in which clauses are propagated. We present here an extended notion of implication graph in which a deduced literal can have several explanations. An extended version of our work can be found in [1]. The paper is organized as follows, definitions and notations are presented in the next section. Section three describes classical conflict analysis. Section four presents our extension, and finally before the conclusion, section five presents some experimental results.

2 Preliminary Definitions and Notations

A *CNF formula* $\mathcal{F}$ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive (x) or negated ($\neg x$) propositional variable. The two literals x and $\neg x$ are called *complementary*. We note $\bar{l}$

H. Kleine Büning and X. Zhao (Eds.): SAT 2008, LNCS 4996, pp. 21–27, 2008.

the complementary literal of l. For a set of literals L, $\bar{L}$ is defined as $\{\bar{l} \mid l \in L\}$. A *unit* clause is a clause with only one literal (called *unit literal*). An *empty clause*, noted $\bot$, is interpreted as false, while an *empty CNF formula*, noted $\top$, is interpreted as true.

The set of variables occurring in $\mathcal{F}$ is noted $V_{\mathcal{F}}$. A set of literals is *complete* if it contains one literal for each variable in $V_{\mathcal{F}}$, and *fundamental* if it does not contain complementary literals. An *interpretation* ρ of a boolean formula $\mathcal{F}$ associates a value $\rho(x)$ to some of the variables $x \in \mathcal{F}$. An interpretation is alternatively represented by a complete and fundamental set of literals, in the obvious way. A *model* of a formula $\mathcal{F}$ is an interpretation ρ that satisfies the formula; noted $\rho \vDash \Sigma$.

The following notations will be heavily used throughout the paper:

- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the literal $\neg x$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \neg x\}$.
- $\mathcal{F}|_x$ will denote the formula obtained from $\mathcal{F}$ by assigning x the truth-value *true*. Formally $\mathcal{F}|_x = \{c \mid c \in \mathcal{F}, \{x, \neg x\} \cap c = \emptyset\} \cup \{c \backslash \{\neg x\} \mid c \in \mathcal{F}, \neg x \in c\}$. This notation is extended to interpretations: given an interpretation $\rho = \{x_1, \ldots, x_n\}$, we define $\mathcal{F}|_\rho = (\ldots((\mathcal{F}|_{x_1})|_{x_2})\ldots|_{x_n})$.
- $\mathcal{F}^*$ denotes the formula $\mathcal{F}$ closed under unit propagation, defined recursively as follows: (1) $\mathcal{F}^* = \mathcal{F}$ if $\mathcal{F}$ does not contain any unit clause, (2) $\mathcal{F}^* = \bot$ if $\mathcal{F}$ contains two unit-clauses $\{x\}$ and $\{\neg x\}$, (3) otherwise, $\mathcal{F}^* = (\mathcal{F}|_x)^*$ where x is the literal appearing in a unit clause of $\mathcal{F}$.

Let us now introduce some notations and terminology on SAT solvers based on the DPLL backtrack search procedure [3]. At each node the assigned literals (decision literal and the propagated ones) are labeled with the same *decision level* starting from 1 and increased at each branching. The current decision level is the highest decision level in the assignment stack. After backtracking, some variables are unassigned, and the current decision level is decreased accordingly. At level i, the current partial assignment ρ can be represented as a sequence of decision-propagations of the form $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ where the first literal x_k^i corresponds to the decision literal x_k assigned at level i and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ represents a propagated (unit) literal at level i. Let $x \in \rho$, we note $l(x)$ the assignment level of x, $d(\rho, i) = x$ if x is the decision literal assigned at level i. For a given level i, we define ρ^i as the projection of ρ to literals assigned at a level $\leq i$.

3 Conflict Analysis Using Implication Graphs

Implication graphs capture the variable assignments ρ made during the search, both by branching and by propagation. This representation is a convenient way to analyze conflicts. In classical SAT solvers, whenever a literal y is propagated, we keep a reference to the clause at the origin of the propagation of y, which we note $\overrightarrow{cla}(y)$. The clause $\overrightarrow{cla}(y)$ is in this case of the form $(x_1 \vee \cdots \vee x_n \vee y)$ where every literal x_i is false under the current partial assignment ($\rho(x_i) = \textit{false}, \forall i \in 1..n$), while $\rho(y) = \textit{true}$. When a literal y is not obtained by propagation but comes from a decision, $\overrightarrow{cla}(y)$ is undefined, which we note for convenience $\overrightarrow{cla}(y) = \bot$.

When $\overrightarrow{cla}(y) \neq \bot$, we denote by $exp(y)$ the set $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, called set of *explanations* of y. In other words if $\overrightarrow{cla}(y) = (x_1 \vee \cdots \vee x_n \vee y)$, then the explanations are the literals $\overline{x_i}$ with which $\overrightarrow{cla}(y)$ becomes the unit clause $\{y\}$. Note that for all i we have $l(\overline{x_i}) \leq l(y)$, i.e., all the explanations of the deduction come from a level at most as high. When $\overrightarrow{cla}(y)$ is undefined we define $exp(y)$ as the empty set. The explanations can alternatively be seen as an implication graph, in which the set of predecessors of a node corresponds to the set of explanations of the corresponding literal:

Definition 1 (Implication Graph). *Let $\mathcal{F}$ be a CNF formula, ρ a partial ordered interpretation, and let exp denote a choice of explanations for the deduced literals in ρ. The implication graph associated to $\mathcal{F}$, ρ and exp is $(\mathcal{N}, \mathcal{E})$ where:*

- *$\mathcal{N} = \rho$, i.e. there is exactly one node for every literal, decision or implied;*
- *$\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in exp(y)\}$*

Example 1. $\mathcal{G}^{\rho}_{\mathcal{F}}$, shown in Figure 1 and restricted to plain arcs is an implication graph for the formula $\mathcal{F}$ and the partial assignment ρ given below : $\mathcal{F} \supseteq \{c_1, \ldots, c_9\}$

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

$\rho = \{\langle \ldots \neg x^1_6 \ldots \neg x^1_{17} \rangle \langle (x^2_8) \ldots \neg x^2_{13} \ldots \rangle \langle (x^3_4) \ldots x^3_{19} \ldots \rangle \ldots \langle (x^5_{11}) \ldots \rangle\}$. The current decision level is 5.

We consider ρ, a partial assignment such that $(\mathcal{F}|_\rho)* = \bot$ and $\mathcal{G}^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E})$ the associated implication graph. Assume that the current decision level is m. As a conflict is reached, then $\exists x \in$ st. $\{x, \neg x\} \subset \mathcal{N}$ and $l(x) = m$ or $l(\neg x) = m$. Conflict analysis is based on applying resolution from the top to the bottom of the implication graph using the different clauses of the form $(\overline{exp(y)} \vee y)$ implicitly encoded at each node $y \in \mathcal{N}$. We call this process a conflict resolution proof. More formally,

Definition 2 (Asserting clause). *A clause c of the form $(\alpha \vee x)$ is called an asserting clause iff $\rho(c) =$ false, $l(x) = m$ and $\forall y \in \alpha, l(y) < l(x)$. x is called asserting literal, which we note in short $\mathcal{A}(c)$. We can define $jump(c) = max\{l(\neg y) \mid y \in \alpha\}$.*

Definition 3 (Conflict resolution proof). *A conflict resolution proof π is a sequence of clauses $\langle \sigma_1, \sigma_2, \ldots \sigma_k \rangle$ satisfying the following conditions :*

1. *$\sigma_1 = \eta[x, \overrightarrow{cla}(x), \overrightarrow{cla}(\neg x)]$, where $\{x, \neg x\}$ is the conflict.*
2. *σ_i, for $i \in 2..k$, is built by selecting a literal $y \in \sigma_{i-1}$ for which $\overrightarrow{cla}(\overline{y})$ is defined. We then have $y \in \sigma_{i-1}$ and $\overline{y} \in \overrightarrow{cla}(\overline{y})$: the two clauses resolve. The clause σ_i is defined as $\eta[y, \sigma_{i-1}, \overrightarrow{cla}(\overline{y})]$;*
3. *σ_k is, moreover an asserting clause.*

It is called elementary iff $\nexists i < k$ s.t. $\langle \sigma_1, \sigma_2, \ldots \sigma_i \rangle$ is also a conflict resolution proof.

4 Extended Implication Graph

In modern SAT solvers, clauses containing a literal x that is implied at the current level are essentially ignored by the propagation. More precisely, because the solver does not maintain the information whether a given clause is satisfied or not, a clause containing x may occasionally be considered by the propagation, but only when another literal y of the clause becomes false. When this happens the solver typically skips the clause. However, in cases where x is true *and all the other literals are false*, an "arc" was revealed for free that could as well be used to extend the graph. Such arcs are those we propose to use in our extension.

To explain our idea let us consider, again, the formula $\mathcal{F}$ and the partial assignments given in the example 1. We define a new formula $\mathcal{F}'$ as follow : $\mathcal{F}' \supseteq \{c_1, \ldots, c_9\} \cup \{c_{10}, c_{11}, c_{12}\}$ where $c_{10} = (\neg x_{19} \vee x_8), c_{11} = (x_{19} \vee x_{10})$ and $c_{12} = (\neg x_{17} \vee x_{10})$.

The three added clauses are satisfied under the instantiation ρ. c_{10} is satisfied by x_8 assigned at level 2, c_{11} is satisfied by x_{19} at level 3, and c_{12} is satisfied by $\neg x_{17}$ at level 1. This is shown in the extended implication graph (see Figure 1) by the doted edges. Let us now illustrate the usefulness of our proposed extension. Let us consider again the the asserting clause Δ_1 corresponding to the classical first UIP. We can generate the following strong asserting clause: $c_{13} = \eta[x_8, \Delta_1, c_{10}] = (x^1_{17} \vee \neg x^3_{19} \vee x^5_{10})$, $c_{14} = \eta[x_{19}, c_{13}, c_{11}] = (x^1_{17} \vee x^5_{10})$ and $\Delta^s_1 = \eta[x_{17}, c_{14}, c_{12}] = x^5_{10}$. In this case we backtrack to the level 0 and we assign x_{10} to *true*. Indeed $\mathcal{F}' \models x_{10}$.

As we can see Δ^s_1 subsumes Δ_1. If we continue the process we also obtain other strong asserting clauses $\Delta^s_2 = (\neg x^3_4 \vee x^5_2)$ and $\Delta^s_3 = (\neg x^3_4 \vee x^2_{13} \vee x^1_6 \vee \neg x^5_{11})$ which subsume respectively Δ_2 and Δ_3. This first illustration gives us a new way to minimize the size of the asserting clauses.

If we take a look to the clauses used in the implication graph $\mathcal{G}^\rho_{\mathcal{F}}$ (plain edges) all have the following properties: (1) $\forall x \in \mathcal{N}$ the clause $c = (\overline{exp(x)} \vee x)$ is satisfied by only one literal i.e. $\rho(x) = \textit{true}$ and $\forall y \in exp(x)$, we have $\rho(y) = \textit{true}$ and (2) $\forall y \in exp(x), l(\neg y) \leq l(x)$. Now in the extended implication graph, the added clauses satisfy property (1) and, in addition, the property (2') $\exists y \in exp(x)$ st. $l(\neg y) > l(x)$.

Let us now explain briefly how the extra arcs can be computed. Usually unit propagation does not keep track of implications from the satisfiable sub-formula. In this extension the new implications (deductions) are considered. For instance in the previous

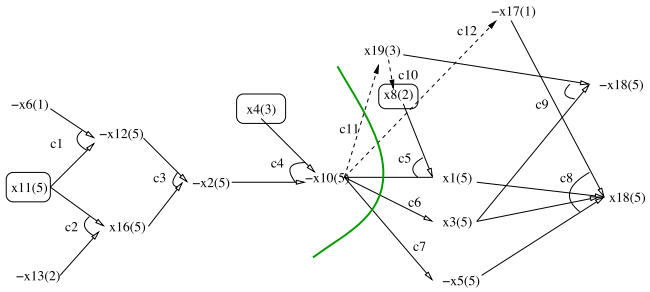


Fig. 1. Implication graph / extended implication graph

example, when we deduce x_{19} at level 3, we "rediscover" the deduction x_8 (which was a choice (decision literal) at level 2). Our proposal keeps track of these re-discoveries.

Before introducing the formal definition of our extended Implication Graph, we introduce the concept of inverse implication (inverse edge).

We maintain additionally to the classical clause $\overrightarrow{cla}(x)$ a new clause $\overleftarrow{cla}(x)$ of the form $(x \vee y_1 \vee \cdots \vee y_n)$. This clause is selected so that $\rho(y_i) = \mathit{false}$ for $i \in 1..n$; $\rho(x) = \mathit{true}$; and $\exists i.\ l(y_i) > l(x)$. This clause can be undefined in some cases (which we note $\overleftarrow{cla}(x) = \perp$). Several clauses of this form can be found for each literal, in which case one is selected arbitrarily: one can choose to consider the first one in the ordering. (It is easy to define a variant where we would take into account all of them, in which case $\overleftarrow{cla}(x)$ is a set of clauses; but we won't develop this variant).

We denote by $\overleftarrow{exp}(x)$ the set $\{\overline{y} \mid y \in \overleftarrow{cla}(x) \setminus \{x\}\}$, and, for clarity, by $\overrightarrow{exp}(x)$ the set that was previously noted exp. An extended implication graph is defined as follows (note that this graph is now not acyclic in general):

Definition 4 (Extended Implication Graph). *Let $\mathcal{F}$ be a CNF formula and ρ an ordered partial interpretation. We define the extended implication Graph associated to $\mathcal{F}$ and ρ as $\mathcal{G}s^{\rho}_{\mathcal{F}} = (\mathcal{N}, \mathcal{E} \cup \mathcal{E}')$ where, $\mathcal{N} = \rho$, $\mathcal{E} = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overrightarrow{exp}(y)\}$ and $\mathcal{E}' = \{(x, y) \mid x \in \rho, y \in \rho, x \in \overleftarrow{exp}(y)\}$*

4.1 Learning to Back-Jump : A First Extension

In this section, we describe a first possible extension of CDCL approach using extended implication graph. Our approach makes an original use of inverses arcs to back-jump farther, i.e. to improve the back-jumping level of the classical asserting clauses.

Let us illustrate the main idea behind our proposed extension. Our approach works in three steps. In the first step (1) : an asserting clause, say $\sigma_1 = (\neg x^1 \vee \neg y^3 \vee \neg z^7 \vee \neg a^9)$ is learnt using the usual learning scheme where 9 is the current decision level. As $\rho(\sigma_1) = false$, usually we backtrack to level $jump(\sigma_1) = 7$. In the second step (2): our approach aims to eliminate the literal $\neg z^7$ from σ_1 using the new arcs of the extended graph. Let us explain this second and new processing. Let $c = (z^7 \vee \neg u^2 \vee \neg v^9)$ such that $\rho(z) = true$, $\rho(u) = true$ and $\rho(v) = true$. The clause c is an inverse arc i.e. the literal z assigned at level 7 is implied by the two literals u and v respectively assigned at level 2 and 9. From c and σ_1, a new clause $\sigma_2 = \eta[z, c, \sigma_1] = (\neg x^1 \vee \neg u^2 \vee \neg y^3 \vee \neg v^9 \vee \neg a^9)$ is generated. We can remark that the new clause σ_2 contains two literals from the current decision level 9. In the third step (3), using classical learning, one can search from σ_2 for another asserting clause σ_3 with only one literal from the current decision level. Let us note that the new asserting clause σ_3 might be worse in terms of back-jumping level. To avoid this main drawback, the inverse arc c is chosen if the two following conditions are satisfied : i) the literals of c assigned at the current level (v^9) has been already visited during the first step and ii) all the other literals of c are assigned before the level 7 i.e. level of z. In this case, we guaranty that the new asserting clause satisfies the following property : $jump(\sigma_3) \leq jump(\sigma_1)$. Moreover, the asserting literal of σ_3 is $\neg a$.

One can iterate the previous process on the new asserting clause σ_3 to eliminate the literals of σ_3 assigned at level $jump(\sigma_3)$ (for more details see [1]).

5 Experiments

Our extended learning scheme can be crafted to any CDCL based solver. See [1] for details. The experimental results reported in this section are obtained on a Xeon 3.2 GHz (2 GB RAM) and performed on a large panel of SAT instances (286) coming from SAT RACE2006 and SAT07 (industrial). All instances are simplified by the satellite preprocessor [4]. Time limit is set to 1800 seconds and results are reported in seconds. We implement our proposed extension to Minisat [5] and Rsat [8] and make a comparison between original solvers and extended ones (called Minisat^E and Rsat^E). Figure 2 shows the $time$ (t) (figure on the left-hand side) and the $cumulated\ time$ (ct) (figure on the right hand side) needed to solve a given number of instances ($nb\ instances$). t and ct represent respectively the number of instances with running time less than t seconds and the number of solved instances if we consider that all the instances are run sequentially within a time limit of t seconds. This global view clearly shows that as t or ct increase the extended versions solve more instances than the original ones.

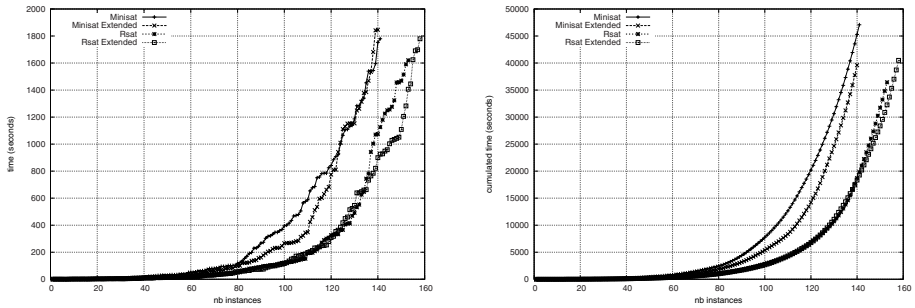


Fig. 2. Time and cumulated time for solving a given number of instances

6 Conclusion

In this paper, we have proposed a generalized framework for conflict analysis. This generalization is obtained by an original extension of the classical implication graph. This extension is obtained by considering clauses that come from the satisfiable part of the formula. Several learning schemes can be defined from this extension. The first extension of learning that improves the classical asserting clauses in term of back-jumping level shows the great potential of the new framework. Despite the different restrictions, our approach achieves interesting improvements of the SAT solvers (Rsat and MiniSat).

References

1. Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., Sais, L.: A Generalized Framework for Conflict Analysis. Microsoft Research, MSR-TR-2008-34 (2008)
2. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. JAIR 22, 319–351 (2004)

3. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)
4. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
5. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, Springer, Heidelberg (2004)
6. Marques-Silva, J.: Personal communication
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of (DAC 2001), pp. 530–535 (2001)
8. Pipatsrisawat, K., Darwiche, A.: Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA (2007)

Learning in Local Search

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, and Lakhdar SAÏS.

In *21st International Conference on Tools with Artificial Intelligence(ICTAI'09)*, pages 417–424, 2009.

Learning in local search *

Gilles Audemard Jean-Marie Lagniez Bertrand Mazure Lakhdar Saïs
Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens
{audemard,lagniez,mazure,sais}@cril.fr

Abstract

In this paper a learning based local search approach for propositional satisfiability is presented. It is based on an original adaptation of the conflict driven clause learning (CDCL) scheme to local search. First an extended implication graph for complete assignments of the set of variables is proposed. Secondly, a unit propagation based technique for building and using such implication graph is designed. Finally, we show how this new learning scheme can be integrated to the state-of-the-art local search solver WSAT. Interestingly enough, the obtained local search approach is able to prove unsatisfiability. Experimental results show very good performances on many classes of SAT instances from the last SAT competitions.

1 Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is a central issue in many computer science and artificial intelligence domains, like e.g. theorem proving, planning, non-monotonic reasoning, VLSI correctness checking. These last two decades, many approaches have been proposed to solve large SAT instances, based on logically complete or incomplete. Both local-search techniques (e.g. [23, 22, 13]) and elaborate variants of the Davis-Putnam-Loveland-Logemann DPLL procedure [5] (e.g. [19, 7]), called modern SAT solvers, can now solve many families of hard SAT instances. These two kinds of approaches present complementary features and performances. Modern SAT solvers are particularly efficient on the industrial SAT category while local search is better on random SAT instances. Consequently, enhancing the performances of local search approaches to the level of modern modern SAT solvers on the industrial category is really an important challenge. This challenge is stated by Bart Selman *et al.* in 1997 [24] (challenge number 6 "Improve stochastic local search on structured problems by efficiently handling variable dependencies"). Another important issue (challenge number 5) also identified in [24], is to design a practical stochastic local search procedure for proving unsatisfiability. The goal of this work, is to make a step towards the resolution of these two challenges. Our aim is to enhance the performances of SLS techniques on industrial SAT instances and to make such techniques able to prove unsatisfiability.

Let us first recall that some attempts towards these directions have been made recently. In [20] functional dependencies recognized using Ostrowski et al approach [12] have been exploited in local search based techniques leading to interesting improvements particularly on crafted SAT instances (e.g. parity 32 instances). In [21, 3] new local search for proving unsatisfiability have been proposed. In [4], authors propose a stochastic local search solver which add resolvents between two clauses in order to leave local minimum. Such method has been improved by [8, 25]. However, all these different approaches are still premature and progress is needed for solving both challenges.

To go further in this direction, we propose to integrate learning from conflict, one of the most important component behind the efficiency of modern SAT solvers, to local search techniques. However, one of the main difference between DPLL-like and local search techniques that make such adaptation very challenging rises in the way search space is explored by both approaches. In DPLL one searches among partial assignments whereas in SLS search is done on complete assignments. Even if such difference is important, the two search paradigms actually admit many common features. One can connect for example the activity based heuristics (VSIDS) with weighting constraint as done in the break-out local search method [18], restarts in modern SAT solvers with tries in WSAT like algorithms.

In this paper, we propose to improve the methods proposed in [4, 8, 25] by integrating Conflict Driven Clause Learning (CDCL) with implication graph [17, 26] to the

*supported by ANR UNLOC project ANR08-BLAN-0289-01

stochastic local search framework. The goal is twofold. First, similarly to [4, 8, 25], we exploit such learning component as a strategy to escape from local minima. However our approach is more general as the conflict clause is generated using an implication graph, whereas in [4, 8, 25], such a clause is obtained using only one resolution step between two clauses. Secondly, like previous methods, the addition of new learnt clauses makes the local search solver able to prove unsatisfiability.

The rest of this paper is organized as follows. In section 2, after the introduction of some preliminary definitions and notations, local search algorithms and classical SAT conflict analysis are presented. In section 3, we describe our extension of the implication graph to complete assignments and introduce a unit propagation based approach for building such graph. Finally, we integrate such conflict analysis in WSAT-like algorithm [23]. In section 4 experimental results of our proposed approach are presented before concluding.

2 Preliminary definitions and technical background

2.1 Definitions

Let us give some necessary definitions and notations. Let $V = \{x_1...x_n\}$ be a set of boolean variables, a literal ℓ is a variable x_i or its negation $\overline{x_i}$. A clause is a disjunction of literals $c_i = (\ell_1 \vee \ell_2 ... \vee \ell_{n_i})$. A unit clause is a clause with only one literal. A formula Σ is in conjunctive normal form (CNF) if it is a conjunction of clauses $\Sigma = (c_1 \wedge c_2 ... \wedge c_m)$. The set of literals appearing in Σ is denoted $\mathcal{V}_\Sigma$. An interpretation $\mathcal{I}$ of a formula Σ associates a value $\mathcal{I}(x)$ to variables in the formula. An interpretation is *complete* if it gives a value to each variable $x \in \mathcal{V}_\Sigma$, otherwise it is said *partial*. A clause, a CNF formula and an interpretation can be conveniently represented as sets. A *model* of a formula Σ, denoted $\mathcal{I} \models \Sigma$, is an interpretation $\mathcal{I}$ which satisfies the formula Σ i.e. satisfies each clause of Σ. Then, we can define the SAT decision problem as follows: is there an assignment of values to the variables so that the CNF formula Σ is satisfied?
Let us introduce some additional notations.

- An empty clause is represented by $\bot$ and is unsatisfiable;
- the negation of a set of literals $\Gamma = \{\ell_1, \ell_2, \ldots, \ell_n\}$ is denoted $\overline{\Gamma}$ ans is equal to $\{\overline{\ell_1}, \overline{\ell_2}, \ldots, \overline{\ell_n}\}$;
- $\Sigma_{|_\ell}$ denotes the formula Σ simplified by the assignment of the literal ℓ to true. This notation is extended to interpretations: Let $\mathcal{P} = \{\ell_1, ..., \ell_n\}$ be an interpretation, $\Sigma_{|_\mathcal{P}} = (...(\Sigma|_{\ell_1})...|_{\ell_n})$;
- Σ^* denotes the formula Σ simplified by unit propagation;
- $\models_*$ denotes logic deduction by unit propagation: $\Sigma \models_* \ell$ means that the literal ℓ is deducted by unit propagation from Σ i.e. $\bot \in (\Sigma \wedge \overline{\ell})^*$. One notes $\Sigma \models_* \bot$ if the formula is unsatisfiable by unit propagation;
- $\eta[x, c_i, c_j]$ denotes the *resolvent* between a clause c_i containing the literal x and c_j a clause containing the opposite literal $\overline{x}$. In other words $\eta[x, c_i, c_j] = c_i \cup c_j \backslash \{x, \overline{x}\}$. A resolvent is called *tautological* when it contains opposite literals.

2.2 Local Search Algorithms

Local search algorithms for SAT problems use a stochastic walk over complete interpretations of Σ. At each *step* (or *flip*), they try to reduce the number of unsatisfiable clauses (usually called a descent). The next complete interpretation is chosen among the neighbours of the current one (they differ only on one literal value). A local minimum is reached when no descent is possible. One of the key point of stochastic local search algorithms is the method used to escape from local minimum. For lack of space, we cannot provide a general algorithm of local search solver. However, a modified version can be seen in algorithm 1. For more details, the reader will refer to [14].

2.3 Conflict Analysis and Implication Graph

Now, we introduce a fundamental data structure, called implication graph, used by complete CDCL solvers (*Conflict Driven Clause Learning*) for conflict analysis, nogood deduction and backjumping. Some of the following notations have been introduced in [2].

A typical branch of a CDCL solver can be seen as sequences of decision-propagation. At decision level i, the current partial interpretation $\mathcal{I}$ is of the form $\langle (x_k^i), x_{k_1}^i, x_{k_2}^i, \ldots, x_{k_{n_k}}^i \rangle$ where the first literal x_k^i corresponds to the decision literal x_k assigned at level i and each $x_{k_j}^i$ for $1 \leq j \leq n_k$ corresponds to a propagated (unit) literal. Whenever a literal y is propagated, we keep a reference to the clause at the origin of the propagation of y, which we denote $\overrightarrow{cla}(y)$. Of course It can exist more than one such clause, however we are taking into account only one of them, usually the first encountered one. The clause $\overrightarrow{cla}(y)$ has, in this case, the form $(x_1 \vee \cdots \vee x_n \vee y)$ where every literal x_i is false under the current partial assignment ($\mathcal{I}(x_i) = false, \forall i \in 1 \ldots n$), while $\mathcal{I}(y) = true$. When a literal y is not obtained by propagation but comes from a decision, $\overrightarrow{cla}(y)$ is undefined, which we note for convenience $\overrightarrow{cla}(y) = \bot$. When $\overrightarrow{cla}(y) \neq \bot$, we denote by $exp(y)$

the set $\{\overline{x} \mid x \in \overrightarrow{cla}(y) \setminus \{y\}\}$, called set of *explanations* of y. In other words, if $\overrightarrow{cla}(y) = (x_1 \vee \cdots \vee x_n \vee y)$, then the explanations are the literals $\overline{x_i}$ with which $\overrightarrow{cla}(y)$ becomes the unit clause $\{y\}$. When $\overrightarrow{cla}(y)$ is undefined we define $exp(y)$ as the empty set.

The implication graph is a directed acyclic graph, it allows a representation of decision-propagations sequences. In such a graph, each vertex is associated to a literal and incoming edges of a vertex are the set of its explanations. Formally, we have:

Definition 1 (Implication graph) *Let Σ be a CNF formula and $\mathcal{I}_p$ be a partial interpretation. An implication graph associated to Σ, $\mathcal{I}_p$ and exp is $\mathcal{G}_\Sigma^{\mathcal{I}_p} = (\mathcal{N}, \mathcal{A})$ where:*

1. *$\mathcal{N} = \{x | x \in \mathcal{I}_p\}$, i.e. there is exactly one node for every literal, decision or implied;*
2. *$\mathcal{A} = \{(x, y) | x \in \mathcal{I}_p, y \in \mathcal{I}_p, x \in exp(y)\}$.*

Example 1 *Let $\Sigma = \{\phi_1, ..., \phi_{11}\}$ be a CNF formula such that.*

$\phi_1 : (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ $\quad$ $\phi_2 : (x_1 \vee \overline{x_4} \vee \overline{x_5})$
$\phi_3 : (x_2 \vee \overline{x_1})$ $\quad$ $\phi_4 : (x_4 \vee \overline{x_7} \vee \overline{x_6})$
$\phi_5 : (x_3 \vee \overline{x_5})$ $\quad$ $\phi_6 : (x_5 \vee \overline{x_7})$
$\phi_7 : (x_6 \vee \overline{x_8})$ $\quad$ $\phi_8 : (x_7 \vee \overline{x_8})$
$\phi_9 : (x_8 \vee \overline{x_4})$ $\quad$ $\phi_{10} : (x_1 \vee \overline{x_8})$
$\phi_{11} : (x_7 \vee \overline{x_9})$

Let $\mathcal{I}_p$ be the following partial interpretation $\mathcal{I}_p = \{\langle(x_2^1)\rangle\langle(x_6^2)\rangle\langle(x_9^3)\ x_7^3\ x_4^3\ x_5^3\ x_3^3\ x_1^3\ \overline{x_1^3}\rangle\}$. The Current decision level is 3 and $\Sigma_{|\mathcal{I}_p} \models_* \bot$. Figure 1 represents the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}_p}$ associated to Σ, $\mathcal{I}_p$ and exp (the set of explanations).

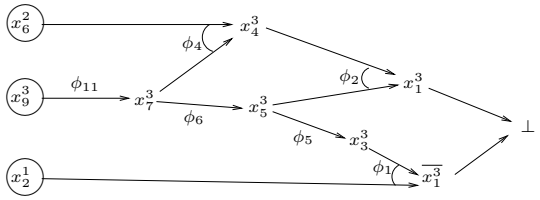


Figure 1. Implication graph (example 1)

As mentioned above, an implication graph allows nogoods extraction. These nogoods are built by a traversal of the implication graph starting from the top (the falsified clause). Different kinds of nogoods can be generated. One of the most used and efficient learning scheme is the first UIP (Unique Implication Point). In the following example, we show a construction of a first UIP. For more details and a formal presentation, the reader can refer to [17, 26].

Example 2 *Consider again example 1 and the same implication graph (Figure 1) associated to interpretation $\mathcal{I}_p$. To generate the first UIP, we perform resolution (starting from the conflict) between clauses encoded in the implication graph (implications):*

- $\beta_1 = \eta(x_1^3, \phi_1, \phi_2) = \overline{x_2^1} \vee \overline{x_3^3} \vee \overline{x_4^3} \vee \overline{x_5^3}$
- $\beta_2 = \eta(x_3^3, \beta_1, \phi_5) = \overline{x_2^1} \vee \overline{x_4^3} \vee \overline{x_5^3}$
- $\beta_3 = \eta(x_5^3, \beta_2, \phi_6) = \overline{x_2^1} \vee \overline{x_4^3} \vee \overline{x_7^3}$
- $\beta = \eta(x_4^3, \beta_3, \phi_4) = \overline{x_2^1} \vee \overline{x_6^2} \vee \overline{x_7^3}$

The clause β contains only one literal from the last decision level (here level 3). Then, the process ends with the clause β, usually called an asserting clause. The literal $\overline{x_7^3}$ is an asserting literal, and the node x_7 in the implication graph (see figure 1) is the first UIP.

3 Local search and conflict analysis

3.1 Conflict graph definition

In section 2.2, we mentioned that one of the key points behind the efficiency of local search algorithms is undoubtedly the strategy used to escape from local minima. In [4, 8, 25], authors proposed to add clauses obtained by resolution in order to leave such minimum. We propose to improve such a strategy by exploiting the implication graph built from a complete interpretation to generate and add nogoods to the clauses data base.

However, in local search framework, one has to deal with complete interpretations. Defining an implication graph in this case is clearly challenging. Indeed, there is no notion of levels or unit propagated literals. Furthermore, a complete interpretation can falsify more than one clause. In this section, we propose a new definition of implication graphs in stochastic local search framework. Before, we give some necessary definitions. Let us consider a CNF formula Σ and a complete interpretation $\mathcal{I}_c$. We say that the literal ℓ satisfies (resp. falsifies) a clause $\beta \in \Sigma$ under $\mathcal{I}_c$ if $\ell \in \mathcal{I}_c \cap \{x \mid x \in \beta\}$ (resp. $\ell \in \mathcal{I}_c \cap \{x \mid \overline{x} \in \beta\}$). We note $\mathcal{L}^+_{\mathcal{I}_c}(\beta)$ (resp. $\mathcal{L}^-_{\mathcal{I}_c}(\beta)$), the set of literals satisfying (resp. falsifying) a clause β under $\mathcal{I}_c$. The following definitions were introduced in [11].

Definition 2 (once-satisfied clause) *A clause β is said* once-satisfied *by an interpretation $\mathcal{I}_c$ on literal z if $\mathcal{L}^+_{\mathcal{I}_c}(\beta) = \{z\}$.*

Definition 3 (critical and linked clauses) *Let $\mathcal{I}_c$ be a complete interpretation. A clause α is critical w.r.t. $\mathcal{I}_c$ if $|\mathcal{L}^+_{\mathcal{I}_c}(\alpha)| = 0$ (α is falsified) and $\forall \ell \in \alpha$, $\exists \alpha' \in \Sigma$ with*

$\overline{\ell} \in \alpha'$ *and* α' *is an once-satisfied clause. Clauses* α' *are linked to* α *for the interpretation* $\mathcal{I}_c$*.*

Example 3 *Let* $\Sigma = (\overline{a} \vee \overline{b} \vee \overline{c}) \wedge (a \vee \overline{b}) \wedge (b \vee \overline{c}) \wedge (c \vee \overline{a})$ *be a formula and* $\mathcal{I}_c = \{a, b, c\}$ *an interpretation. The clause* $\alpha_1 = (\overline{a} \vee \overline{b} \vee \overline{c})$ *is critical. The other clauses of* Σ *are linked to* α_1 *for* $\mathcal{I}_c$*.*

Now, we can define conflict graph for complete interpretations.

Definition 4 (Conflict graph on z) *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation falsifying* Σ*. Consider two clauses of* Σ*,* $\beta = \{\beta_1, \ldots, \beta_k, z\}$ *falsified by* $\mathcal{I}_c$ *and* $\gamma = \{\gamma_1, \ldots, \gamma_l, \overline{z}\}$ *once-satisfied on* $\overline{z}$*, the conflict graph* $\mathcal{G}^z_{\Sigma,\mathcal{I}_c} = (\mathcal{N}, \mathcal{A})$ *is constructed in the following way:*

1. - $\{z, \overline{z}, \bot\} \subseteq \mathcal{N}$ *;*
   - $\{\overline{\gamma_1}, \ldots, \overline{\gamma_l}\} \subseteq \mathcal{N}$ *;*
   - $\{\overline{\beta_1}, \ldots, \overline{\beta_k}\} \subseteq \mathcal{N}$ *;*
2. - $\{(z, \bot), (\overline{z}, \bot)\} \subseteq \mathcal{A}$ *;*
   - $\{(\overline{\beta_1}, z), \ldots, (\overline{\beta_k}, z)\} \subseteq \mathcal{A}$ *;*
   - $\{(\overline{\gamma_1}, \overline{z}), \ldots, (\overline{\gamma_k}, \overline{z})\} \subseteq \mathcal{A}$ *;*
3. $\forall x \in \mathcal{N}$*, if* $x \neq z$ *and* $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \bot$ *then* $\overline{\alpha} \vee x \in \Sigma$ *is once-satisfied on* x*.*

Since for a given literal z, clauses like β and γ are not unique, then the conflict graph is not unique too.

Example 4 *Consider again example 1. Let* $\mathcal{I}_c$ *be a complete interpretation such that* $\mathcal{I}_c = \{x_1, \ldots, x_9\}$*. Figure 2 represents the conflict graph* $\mathcal{G}^{x_1}_{\Sigma,\mathcal{I}_c}$ *constructed on variable* x_1*.*

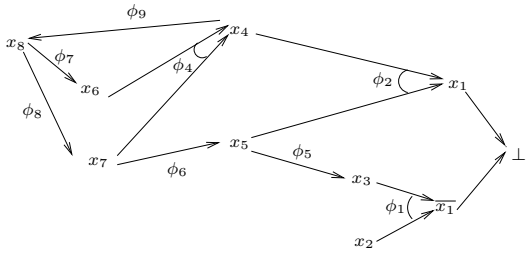


Figure 2. Conflict graph constructed on variable x_1 (example 4)

It is important to note that the existence of such a graph is not ensured. The following proposition gives necessary and sufficient conditions for the construction of the conflict graph.

Proposition 1 *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation. The conflict graph* $\mathcal{G}^x_{\Sigma,\mathcal{I}_c}$ *exists if and only if* x *and* $\overline{x}$ *appear respectively in a falsified clause and in a once-satisfied one.*

Proof The proof of this proposition follows directly from the definition of the conflict graph.

Corollary 1 *Let* $\alpha \in \Sigma$ *be a critical clause.* $\forall x \in \alpha$*, it is possible to construct a conflict graph on* x*.*

Proof By definition of a critical clause $\alpha \in \Sigma$, we have $\forall x \in \alpha, \exists \beta \in \Sigma$ once-satisfied on x. From Proposition 1, it is obvious that $\forall x \in \alpha$, it exists a conflict graph on x.

As shown in [11], in a local minima all falsified clauses are necessarily critical. The corollary 1 ensures that, in this case, it is always possible to build a conflict graph. Then, we can use it in order to leave such local minimum. However, generating such nogoods is not obvious. Indeed, there is not notion of levels, the classical notion of conflict analysis and first UIP can not be extended. Furthermore, conflict graphs can contain cycles. In this case, the resolution step can produce tautological clauses which are obviously useless. To overcome these problems, we propose to transform conflict graph of a complete interpretation into a classical implication graph and then use classical learning to generate relevant nogoods.

3.2 Building Conflict Graph

To overcome problems introduced in previous section, we propose a first method, based on unit propagation. Starting from a complete interpretation $\mathcal{I}_c$, we build a partial interpretation $\mathcal{I}'$ by unit propagation where decision variables have the same value as in $\mathcal{I}_c$. This partial interpretation introduced formally in the following definition will help us to build the conflict graph.

Definition 5 (Derived partial interpretation) *Let* Σ *be a CNF formula,* $\mathcal{I}_c$ *a complete interpretation. The derived partial interpretation of* $\mathcal{I}_c$*, denoted* $\mathcal{I}'$*, is incrementally build as follows:*

- $\mathcal{I}'_0 = \emptyset$ *;*
- $\mathcal{I}'_{i+1} = \mathcal{I}'_i \cup \{(x_{i+1}), x^1_{i+1}, ..., x^k_{i+1}\}$ *such that* $x_{i+1} \in \mathcal{V}_\Sigma \setminus \mathcal{I}'_i$ *and* $\forall j, 1 \leqslant j \leqslant k$ *one has* $\Sigma_{|_{\mathcal{I}'_i \cup \{x_{i+1}\}}} \models_* x^j_{i+1}$ *with* $x^j_{i+1} \in \mathcal{I}_c$ *;*
- $\mathcal{I}' = \mathcal{I}'_i \cup \{(x_{i+1}), x^1_{i+1}, ..., x^l_{i+1}\}$ *such that* $x_{i+1} \in \mathcal{V}_\Sigma \setminus \mathcal{I}'_i$ *and* $\forall j, 1 \leqslant j \leqslant l$ *one has* $\Sigma_{|_{\mathcal{I}'_i \cup \{x_{i+1}\}}} \models_* x^j_{i+1}$ *with* $x^j_{i+1} \in \mathcal{I}_c$ *for* $j \neq l$ *and* $x^l_{i+1} \notin \mathcal{I}_c$*.*

The set of variables associated to the decision literals is called a conflict set. Furthermore, we call conflict variable the one associated to the literal $x \in \mathcal{I}' \setminus \mathcal{I}$. The literal x is also called a conflict literal.

Of course, the choice of the conflict set of variables is a heuristic choice and can lead to different derived partial interpretations.

Example 5 *Let us consider again the CNF Σ of example 1 and the complete interpretation $\mathcal{I}_c = \{x_1, \dots x_9\}$.*
First, considering the decision variables in lexicographic ordering. We have:

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_1), x_2^1, \overline{x_3^1}\}$

Then, the conflict variable is x_3 and the conflict set is limited to $\{x_1\}$.
Now, considering the inverse lexicographic ordering, the partial interpretation is:

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_9^1), x_7^1, x_5^1, x_1^3\}$
- $\mathcal{I}'_2 = \{(x_9^1), x_7^1, x_5^1, x_1^3, (x_8^2), x_6^2, x_4^2, x_1^2, x_2^2, \overline{x_2^2}\}$

the conflict variable is x_2 and the conflict set is $\{x_9, x_8\}$.

Complete interpretation is conflicting, so the derived partial interpretation will differ on, at least, one conflict literal. The following proposition asserts that it exists at least one conflict clause containing this literal. The conflict graph is then built on this literal.

Proposition 2 *Let Σ be a CNF formula, $\mathcal{I}_c$ a conflicting complete interpretation and $\mathcal{I}'$ a derived partial interpretation of $\mathcal{I}_c$. Let x be the conflict literal, then $exp(x) \subseteq \mathcal{I}_c$ and the clause $\overrightarrow{cla}(x)$ is falsified by $\mathcal{I}_c$.*

Proof First, by construction of $\mathcal{I}'$, it is obvious that $exp(x) \subseteq \mathcal{I}_c$. Indeed, suppose that $exp(x) \not\subseteq \mathcal{I}_c$ then $\exists y \in exp(x)$ such that $y \notin \mathcal{I}_c$. By definition $exp(x) \subseteq \mathcal{I}'$, so $y \in \mathcal{I}'$, consequently the literal y is also a conflict literal. By construction of $\mathcal{I}'$, it exists only one conflict literal, then $y = x$. This is impossible, because a propagated literal can not be included in its explanation.

Secondly, we have to prove that $\mathcal{I}_c \not\models \overrightarrow{cla}(x)$. Suppose $\overrightarrow{cla}(x)$ is satisfied by $\mathcal{I}_c$. Note that x is a propagated literal, then it exists an explanation $exp(x)$ and a clause $\overrightarrow{cla}(x) \in \Sigma$ such that $\overrightarrow{cla}(x) = \overline{exp(x)} \vee x$. We know that $exp(x) \subseteq \mathcal{I}_c$, then $\overline{exp(x)} \not\subseteq \mathcal{I}_c$. Since $\overrightarrow{cla}(x)$ is satisfied by $\mathcal{I}_c$, it can be only satisfied by x. This is not possible because $x \notin \mathcal{I}_c$.

The following proposition expresses the fact that all clauses (expect the falsified one) which are used to construct the graph are once-satisfied clauses.

Proposition 3 *Let Σ be a CNF formula, $\mathcal{I}_c$ a complete conflicting interpretation, $\mathcal{I}'$ a derived partial interpretation and x the conflict literal associated to $\mathcal{I}'$. Consider the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$, then $\forall y \in \mathcal{N} \setminus \{x\}$ one has $\overrightarrow{cla}(y) = \bot$ where $\overrightarrow{cla}(y)$ is once-satisfied by $\mathcal{I}'$ on x.*

Proof One need to consider two cases:

1. y is a decision literal, then $\overrightarrow{cla}(y) = \bot$;
2. y is a propagated literal. It exists $\overrightarrow{cla}(y) \in \Sigma$ such that $\overrightarrow{cla}(y) = \overline{exp(y)} \vee y$. By construction of $\mathcal{I}'$, one has $x \notin exp(y)$ and $\mathcal{I}' \setminus \{x\} \subset \mathcal{I}_c$. Since $y \neq x$, one has $\{exp(y), y\} \subseteq \mathcal{I}'$. By transitivity, one obtain $\{exp(y), y\} \subseteq \mathcal{I}_c \setminus \{x\}$. Then, the clause $\overrightarrow{cla}(y)$ is once-satisfied by $\mathcal{I}_c$ on y.

In the proposition 4, we show that the implication graph obtained with the partial derived interpretation can be extended in a conflict graph on the conflict literal. Then, with the help of this implication graph, it is possible to generate nogoods similarly to classical CDCL solvers [7]. Of course, these nogoods will be added to the clauses database.

Proposition 4 *Let Σ be a CNF formula, $\mathcal{I}_c$ a complete interpretation on Σ, $\mathcal{I}'$ a partial derived interpretation and x the conflict literal associated to $\mathcal{I}'$. If $\exists \alpha \in \Sigma$ once-satisfied by $\mathcal{I}_c$ on x, then it is possible to extend the implication graph $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$ associated to $\mathcal{I}'$ to a conflict graph $\mathcal{G}^x_{\Sigma,\mathcal{I}_c} = (\mathcal{N}', \mathcal{A}')$ as follows:*

- $\mathcal{N}' = \mathcal{N} \cup \{y \in \overline{\alpha} \setminus x\} \cup \{\overline{x}, \bot\}$;
- $\mathcal{A}' = \mathcal{A} \cup \{(y, \overline{x}) | y \in \overline{\alpha} \setminus x\} \cup \{(x, \bot), (\overline{x}, \bot)\}$.

Proof Before proving that $\mathcal{G}^x_{\Sigma,\mathcal{I}_c}$ is a valid conflict graph, one has to identify clauses $\beta = \{\beta_1, \dots, \beta_k, z\} \in \Sigma$ falsified by $\mathcal{I}_c$ and $\gamma = \{\gamma_1, \dots, \gamma_l, \overline{z}\} \in \Sigma$ once-satisfied on z for $\mathcal{I}_c$. By hypothesis, clause α is once-satisfied by $\mathcal{I}_c$ on x. Obviously, $\gamma = \alpha$. By proposition 2, one can take $\beta = \overrightarrow{cla}(x)$. Indeed, $x \in \overrightarrow{cla}(x)$ and the clause $\overrightarrow{cla}(x)$ is falsified by $\mathcal{I}_c$. The both clauses used for the construction of the conflict graph are now identified. To prove that $\mathcal{G}^x_{\Sigma,\mathcal{I}_c} = (\mathcal{N}', \mathcal{A}')$ is a conflict graph, one has to verify the following properties (see definition 4):

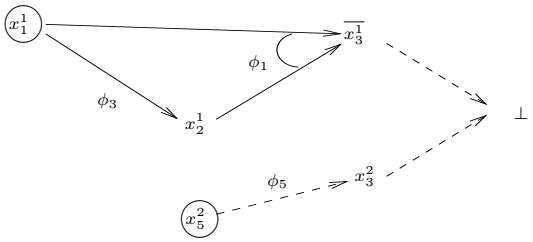
1. - $\{x, \overline{x}, \bot\} \subseteq \mathcal{N}'$. By hypothesis, one has $\{\overline{x}, \bot\} \subseteq \mathcal{N}'$, We only need to prove that $x \in \mathcal{N}'$. We know that $x \in \mathcal{I}'$, by construction of $\mathcal{G}_\Sigma^{\mathcal{I}'}$, then we have $x \subseteq \mathcal{N}$. Since $\mathcal{N} \subseteq \mathcal{N}'$, then $x \in \mathcal{N}'$;

- $\{\overline{\gamma_1}, \ldots, \overline{\gamma_l}\} \subseteq \mathcal{N}'$. We have $\{y \in \overline{\alpha} \setminus x\} \subseteq \mathcal{N}'$ and $\gamma = \alpha$. Then $\{y \in \overline{\gamma} \setminus x\} \subseteq \mathcal{N}'$;
- $\{\overline{\beta_1}, \ldots, \overline{\beta_k}\} \subseteq \mathcal{N}'$. By hypothesis, $\beta = \overrightarrow{cla}(x) = \beta_1 \vee \cdots \vee \beta_k \vee x = \overline{exp}(x) \vee x$. Then, $exp(x) = \{\overline{\beta_1}, \ldots, \overline{\beta_k}\}$. By proposition 2, one has $exp(x) \subseteq \mathcal{I}'$, so $exp(x) \subseteq \mathcal{N}$ (see definition 1). Since $\mathcal{N} \subseteq \mathcal{N}'$, by transitivity, one has $exp(x) \subseteq \mathcal{N}'$;

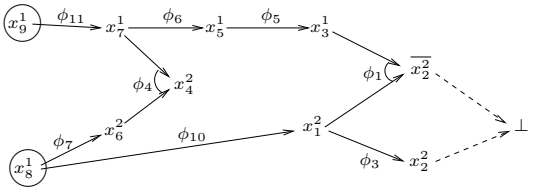
2.
- $\{(x, \bot), (\overline{x}, \bot)\} \subseteq \mathcal{A}'$. By construction ;
- $\{(\overline{\gamma_1}, \overline{x}), \ldots, (\overline{\gamma_k}, \overline{x})\} \subseteq \mathcal{A}'$. By construction ;
- $\{(\overline{\beta_1}, x), \ldots, (\overline{\beta_k}, x)\} \subseteq \mathcal{A}'$. One knows that $exp(x) = \{\overline{\beta_1}, \ldots, \overline{\beta_k}\}$ and $\{exp(x), x\} \subseteq \mathcal{I}'$. By construction of $\mathcal{A}'$ and by definition 1, one has $\{(\overline{\beta_1}, x), \ldots, (\overline{\beta_k}, x)\} \subseteq \mathcal{A} \subseteq \mathcal{A}'$;

3. $\forall x \in \mathcal{N}$, if $x \neq z$ and $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \bot$ then $\overline{\alpha} \vee x \in \Sigma$ and is once-satisfied on x. By proposition 3, $\forall y \in \mathcal{N}$ such that $y \neq x$ and $\overrightarrow{cla}(y) \neq \bot$, one has $\overrightarrow{cla}(y)$ once-satisfied by $\mathcal{I}_c$ on y. By construction of $\mathcal{G}_\Sigma^{\mathcal{I}'}$ and $\mathcal{G}_{\Sigma,\mathcal{I}_c}^x$ the previous property is verified.

Example 6 *Let us take again example 1 and partial derived interpretations obtained in example 5. We can extend implication graphs associated to these interpretations in two conflict graphs depicted in Figures 3.a (lexicographic order) and 3.b (inverse lexicographic order).*



a. lexicographic order



b. inverse lexicographic order

Figure 3. Conflict graph constructed with unit propagation (example 6)

3.3 Implementation

We propose to incorporate the conflict graph defined in previous section inside a WSAT like solver [23]. This is done when a local minima is reached. We name this method CDLS. During the construction of the partial derived interpretation, two cases might occur: Either a conflict is reached during unit propagation (see Figure 3.b) or not (see Figure 3.a). In the former, one uses the resulting implication graph, analyzes the conflict and extracts an assertive clause associated to the first UIP. In the latter, one extends the implication graph of the derived partial interpretation to a conflict graph and, in a similar way, an assertive clause is extracted. In both cases, assertive clauses are added to the clauses database of the formula. And, if the assertive clause is the empty one, the unsatisfiability of the formula is proved. Contrary to classical WSAT algorithm, one can flip a set of variables when a local minima is reached. These are variables whose values differ between complete interpretation and derived partial interpretation.

It is important to note that CDLS is not an hybrid algorithm like [9, 10]. It is a simple stochastic local search method. Unit propagation is only used to build the conflict graph, to analyze conflict graph and to extract nogoods.

Algorithm 1: CDLS

Input: Σ a CNF formula
Output: SAT if Σ is satisfiable, $UNSAT$ if Σ is unsatisfiable, else $UNKNOWN$

for $i \leftarrow 1$ *to* $MaxTries$ **do**
  $\mathcal{I}_c \leftarrow$ completePUInterpretation(Σ);
  for $j \leftarrow 1$ *to* $MaxFlips$ **do**
    if $\mathcal{I}_c \models \Sigma$ **then**
      return SAT;
    $\Gamma = \{\alpha \in \Sigma | \ \mathcal{I}_c \not\models \alpha\}$;
    while $\Gamma \neq \emptyset$ **do**
      $\alpha \in \Gamma$;
      if $\exists x \in \alpha$ *allowing a descent* **then**
        $flip(x)$;
        break;
      else
        $\Gamma \leftarrow \Gamma \setminus \{\alpha\}$;
    if $\Gamma = \emptyset$ **then** `/* local minimum */`
      $\alpha \in \Sigma$ such that $\mathcal{I}_c \not\models \alpha$;
      $\beta \leftarrow conflictAnalysisRL(\Sigma, \mathcal{I}_c, \alpha)$;
      if $\beta = \bot$ **then**
        return UNSAT;
      $\Sigma \leftarrow \Sigma \cup \{\beta\}$;
return $UNKNOWN$;

Algorithm 2: completePUInterpretation

Input: Σ a CNF formula
Output: $\mathcal{I}_c$ a complete interpretation of Σ

1 $\Sigma' \leftarrow \Sigma$; $\mathcal{I}_c \leftarrow \emptyset$;
2 **while** $\Sigma' \neq \emptyset$ **do**
3 $\quad x \in lit(\Sigma')$;
4 $\quad \mathcal{P} \leftarrow \{x\} \cup \{y | \Sigma_{|_x} \models_* y\}$;
5 $\quad$ **foreach** $y \in \mathcal{P}$ **do**
6 $\quad\quad$ **if** $\overline{y} \in \mathcal{P}$ **then**
7 $\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \setminus \{y\}$;
8 $\quad \Sigma' \leftarrow \Sigma'_{|_{\mathcal{P}}}$;
9 $\quad \mathcal{I}_c \leftarrow \mathcal{I}_c \cup \mathcal{P}$;
10 **return** $\mathcal{I}_c$;

Algorithm 3: conflictSet

Input: Σ a CNF; $\mathcal{I}_c$ a complete interpretation; $\alpha \in \Sigma$ a critical clause for $\mathcal{I}_c$.
Output: $\mathcal{C}$ a conflict literals set

1 $\mathcal{C} \leftarrow \emptyset$;
2 **forall** $x \in \overline{\alpha}$ **do**
3 $\quad \beta \in \Sigma$ once-satisfied on x;
4 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\beta \setminus \{x\}\}$;
5 **return** $\mathcal{C}$;

Algorithm CDLS (see Algorithm 1) takes a CNF formula Σ as input and returns three different values (SAT, $UNSAT$ or $UNKNOWN$). It is based on WSAT algorithm. Note that initial complete interpretations are generated using unit propagation (line 2 of Algorithm 1 and Algorithm 2). In this way, local minimas are quickly reached and variables dependencies are taken into account i.e using unit propagation. Whenever a descent is possible, one flips a variable allowing it. When a local minima is reached, we analyze the conflict as explained previously. A nogood β is generated and added to the clause database.

Algorithm 4 is the core or our proposed framework. It starts by selecting conflict variables set which allows to build the derived partial interpretation. This is done by Algorithm 3. It chooses variables in linked clauses to the falsified clause α in order to make the generated partial interpretation nearest to α. Then, this partial interpretation $\mathcal{I}_p$ is constructed (line 4-7). Two cases might occur. In the former, a conflict is reached, classical conflict analysis is applied (line 9) and an assertive literal is flipped. In the latter, one can extracts the partial derived interpretation of $\mathcal{I}_p$ and generates the associated conflict graph (line 12-14 and proposition 4). At this point, conflict analysis can be achieved. All variables with different values in both interpretations are then flipped.

Algorithm 4: conflictAnalysisRL

Input: Σ a CNF; $\mathcal{I}_c$ a complete interpretation; $\alpha \in \Sigma$ a critical clause for $\mathcal{I}_c$.
Output: β a clause built on $\mathcal{V}_\Sigma$

1 $\mathcal{E} \leftarrow conflictSet(\Sigma, \mathcal{I}_c, \alpha)$; $\gamma \leftarrow \emptyset$; $\mathcal{I}_p \leftarrow \emptyset$;
2 **while** $(\gamma = \emptyset)$ *and* $(\mathcal{I}_p \subset \mathcal{I}_c)$ **do**
3 $\quad \mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{I}_p$; $\mathcal{I}_p \leftarrow \mathcal{I}_p \cup \{x\}$ such that $x \in \mathcal{E}$; $\gamma \leftarrow BCP()$;
4 **if** $\gamma \neq \emptyset$ **then** `/* CASE 1 */`
5 $\quad \beta \leftarrow firstUIP(\mathcal{G}_\Sigma^{\mathcal{I}_p})$;
6 $\quad flip(x)$ with x assertive literal ;
7 **else** `/* CASE 2 */`
8 $\quad \mathcal{I}' \leftarrow$ partial interpretation associated to $\mathcal{I}_p$;
9 $\quad y \leftarrow$ conflict literal of $\mathcal{I}'$;
10 $\quad \mathcal{G}^y_{(\Sigma,\mathcal{I}')} \leftarrow$ extended conflict graph $\mathcal{G}_\Sigma^{\mathcal{I}'}$;
11 $\quad \beta \leftarrow firstUIP(\mathcal{G}^y_{(\Sigma,\mathcal{I}')})$;
12 $\quad$ **forall** $x \in \mathcal{I}_p \setminus \mathcal{I}_c$ **do** $flip(x)$;
13 **return** β;

	Crafted		Industrial		Random	
	sat	unsat	sat	unsat	sat	unsat
ADAPTG2	326	0	232	0	1111	0
RSAPS	339	0	226	0	1071	0
WSAT	259	0	206	0	1012	0
CDLS	331	146	412	232	943	0
CLS	235	75	227	102	690	0
MINISAT	402	369	588	414	609	315

Table 1. CDLS **versus some other SAT solvers**

4 Experimental results

Experimental results reported in this section were obtained on a Xeon 3.2 GHz with 2 GByte of RAM. CPU time is limited to 1200 seconds. We compare CDLS to three classical incomplete local search methods, WSAT [23], RSAPS [15] and ADAPTG2 [16] ; we also add CLS the complete local search solver proposed by [8] and MINISAT [7] one of state-of-the-art CDCL solver. Instances used are taken from the last SAT competitions. They are divided into different categories: crafted (1439 instances), industrial (1305) and random (2172). All instances are preprocessed wih SatElite [6]. Indeed, it is well known resolution based preprocessors help, in a lot of cases, local search and complete ones [1].

Table 4 summarizes the obtained results on this large number of instances. For more details on this experimental part, the reader can refer to `http://www.cril.fr/~lagniez/cdls`. For each category and for each solver we report the number of solved instances. Of course, MIN-

ISAT a state-of-the-art CDCL based complete solver is considered here, only to mention the gap between local search based techniques and complete modern SAT solvers on industrial and crafted instances. On random satisfiable instances, local search techniques generally outperform complete techniques.

Let us start to analyze the results obtained on the crafted category. Except for MINISAT, CDLS is very competitive and solves approximately the same number of instances than RSAPS and ADAPTG2. Furthermore, CDLS solves much more instances than WSAT, its built-in solver. Comparing to CLS, our solver is better, it solves more SAT and UNSAT instances. For industrial instances, CDLS solves two times more instances than other stochastic local search solvers and 232 unsatisfiable instances. It outperforms CLS, the other complete local search solver. So, conflict analysis allows to solve efficiently structured SAT and UNSAT instances. Finally, for the random category, we can note that CDLS and CLS are unable to solve unsatisfiable problems. As pointed by MINISAT results, learning is not the good approach to solve random instances.

A summary, our solver CDLS is much more efficient than other local search algorithms. It significantly improves WSAT, its built-in solver and CLS another complete local search approach. Even if MINISAT is the best solver on crafted and industrial instances, these first results are very encouraging and reduce the gap between local search based techniques and DPPL-like complete solvers.

5 Conclusion

In this paper, we propose a new approach to keep out local minimum when dealing with a stochastic local search solver. Our approach extends conflict analysis used by modern SAT solvers. With this extension, we significantly improved stochastic local search solvers. More interestingly, our approach is able to prove inconsistency of many SAT instances, and then it can be seen as an important step to the resolution of the challenge number 5 proposed by Selman et al. at IJCAI 1997. These first results are very promising. Our solver is able to solve a lot of unsatisfiable instances and it achieves interesting improvements of local search based techniques on structured SAT instances. In future works, we plan to improve heuristic choice for the conflict set. We also want to analyze and extract nogoods without the help of unit propagation. Finally, we plan to study other schemes for extracting relevant nogoods.

References

[1] Anbulagan, D.N. Pham, J. Slaney, and A. Sattar. Boosting sls performance by incorporating resolution-based preprocessor. In *proceedings of the workshop LSCS (in conjonction to CP)*, 2006.

[2] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Saïs. A generalized framework for conflict analysis. In *proceedings of SAT*, pages 21–27, 2008.

[3] G. Audemard and L.Simon. Gunsat: A greedy local search algorithm for unsatisfiability. In *Proceedings of IJCAI*, pages 2256–2261, 2007.

[4] B. Cha and K. Iwama. Adding new clauses for faster local search. In *proceedings of AAAI*, pages 332–337, 1996.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of ACM*, 5(7):394–397, 1962.

[6] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[7] N. Een and N. Sörensson. An extensible SAT-solver. In *proceedings of SAT*, pages 502–518, 2003.

[8] H. Fang and W. Ruml. Complete local search for propositional satisfiability. In *proceedings of AAAI*, pages 161–166, 2004.

[9] L. Fang and M. Hsiao. A new hybrid solution to boost SAT solver performance. In *proceedings of DATE*, pages 1307–1313, 2007.

[10] E. Goldberg. A decision-making procedure for resolution-based SAT-solvers. In *proceedings of SAT*, pages 119–132, 2008.

[11] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *proceedings of ECAI*, pages 387–391, 2006.

[12] É. Grégoire, R. Ostrowski, B. Mazure, and L. Saïs. Automatic extraction of functional dependencies. In *proceedings of SAT*, pages 122–132, 2004.

[13] E.A. Hirsch and A. Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematical and Artificial Intelligence*, 43(1):91–111, 2005.

[14] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[15] F. Hutter, D. Tompkins, and H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *proceedings of CP*, pages 233–248, 2002.

[16] Chu Min Li, Wanxia Wei, and Harry Zhang. Combining adaptive noise and look-ahead in local search for sat. In *proceedings of SAT*, pages 121–133, 2007.

[17] J. Marques-Silva and K. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *proceedings of ICCAD*, pages 220–227, 1996.

[18] P. Morris. The breakout method for escaping from local minima. In *proceedings of AAAI*, pages 40–45, 1993.

[19] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *proceedings of DAC*, pages 530–535, 2001.

[20] D. N. Pham, J. Thornton, and A. Sattar. Building structure into local search for SAT. In *Proceedings of IJCAI*, pages 2359–2364, 2007.

[21] S. Prestwich and I. Lynce. Local search for unsatisfiability. In *proceedings of SAT*, pages 283–296, 2006.

[22] B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In *proceedings of AAAI*, pages 46–51, 1993.

[23] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *proceedings of AAAI*, pages 337–343, 1994.

[24] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, pages 50–54, 1997.

[25] H. Shen and H. Zhang. Another complete local search method for SAT. In *proceedings of LPAR*, pages 595–605, 2005.

[26] L. Zhang, C.F. Madigan, M.W. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *proceedings of ICCAD*, pages 279–285, 2001.

SAT graph based representation : A new perspective

Gilles AUDEMARD, Said JABBOUR, and Lakhdar SAÏS.

Journal of Algorithms in Cognition, Informatic and Logic, 63 :17–33, 2008.

Available online at www.sciencedirect.com
ScienceDirect
Journal of Algorithms: Algorithms in Cognition, Informatics and Logic 63 (2008) 17–33
www.elsevier.com/locate/jalgor

SAT graph-based representation: A new perspective

Gilles Audemard [a,b,c,*], Said Jabbour [a,b,c], Lakhdar Sais [a,b,c]

[a] *Université Lille-Nord de France, Artois, F-62307 Lens, France*
[b] *CRIL, F-62307 Lens, France*
[c] *CNRS UMR 8188, F-62307 Lens, France*

Received 22 October 2007
Available online 7 March 2008

Abstract

In this paper a new graph based representation of Boolean formulas in conjunctive normal form (CNF) is proposed. It extends the well-known graph representation of binary CNF formulas (2-SAT) to the general case. Every clause is represented as a set of (conditional) implications and encoded with different edges labeled with a set of literals, called context. This representation admits many interesting features. For example, a path from the node labeled with a literal $\neg x$ to the node labeled with a literal x gives us an original way to compute the condition under which the literal x is implied. Using this representation, we show that classical resolution can be reformulated as a transitive closure on the generated graph. Interestingly enough, using the SAT graph-based representation three original applications are then derived. The first one deals with the 2-SAT strong backdoor set computation problem, whereas in the second one the underlying representation is used to derive hard SAT instances with respect to the state-of-the-art satisfiability solvers. Finally, a new preprocessing technique of CNF formulas which extends the well-known hyper-resolution rule is proposed. Experimental results show interesting improvements on many classes of SAT instances taken from the last SAT competitions.

Keywords: Satisfiability; Graph representation; Backdoors; Preprocessing; Benchmarks

1. Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is central in many computer science and artificial intelligence domains, like e.g. theorem proving, planning, non-monotonic reasoning, VLSI correctness checking and knowledge-based verification and validation. During these last two decades, many approaches have been proposed to solve hard SAT instances, based on-logically complete or not-algorithms. Both local-search techniques (e.g. [1]) and elaborate variants of the Davis–Putnam–Loveland–Logemann's DPLL procedure [2] (e.g. [3,4]) can now solve many families of hard huge SAT instances from real-world applications.

Most of the successful complete solvers are based on the backtrack search algorithm called Davis–Putnam–Logemann–Loveland (DPLL) procedure. Such basic algorithm is enhanced with many important pruning techniques such as learning, extended use of Boolean constraint propagation, preprocessing, symmetries breaking, etc. The im-

* Corresponding author at: Université Lille-Nord de France, Artois, F-62307 Lens, France.
E-mail addresses: audemard@cril.fr (G. Audemard), jabbour@cril.fr (S. Jabbour), sais@cril.fr (L. Sais).

0196-6774/$ – see front matter
doi:10.1016/j.jalgor.2008.02.001

pact of those different improvements depends on the kind of instances to be solved. For example, learning is more useful when solving instances encoding real world problems than on random generated ones. Another important aspect for efficient SAT solving concerns the problem encoding. Traditionally, most solvers work on formulas encoded in conjunctive normal form (CNF). However, encoding knowledge under CNF can flatten some structural knowledge that would be more apparent in more expressive propositional logic representation formalisms, and that could prove useful in the resolution step [5–7]. To take benefit from such structural knowledge, recent works have addressed this issue following three different paths of research. The first one uses extended Boolean formulas (nonCNF [7], pseudo-Boolean constraints [8], ...) for problem encoding. Whereas the second one tries to recover and/or to deduce structural knowledge from CNF encoding (symmetries [9], functional dependencies [6], equivalence [10]). Finally, different CNF graph-based presentations have been proposed for modeling (or solving) SAT instances. This last kind of approach follows different motivations: structure visualization [11], CNF decomposition [12], efficient propagation [13], learning (from implication graph) [3,14].

In this paper, a new SAT graph-based representation is proposed. It extends in an original way the well-known graph representation of binary CNF formulas (2-SAT) to the general case. Every clause is represented as a set of possible (conditional) implications and encoded with different edges labeled with a set of literals, called contexts (or conditions). Our proposed representation allows us to extend many interesting features of the 2-SAT polynomial time algorithm. Among them, classical resolution is formulated using the transitive closure of the graph. Paths between nodes labeled with opposite literals lead to a new definition of conditional backbone i.e. the literal is implied under a given context or condition. However, in the general case finding the (minimal) context under which a given literal is implied is computationally intractable.

Clearly, our SAT graph based representation offers a new perspective on the exploitation of the hidden structures of CNF formulas. In this paper, to benefit from such representation, three different applications are proposed. The first one deals with the strong backdoor set computation problem, the second exploits the underlying structure of the graph to derive hard SAT instances, and the third one is a preprocessing technique extending the well known hyper resolution principle.

The paper is organized as follows. After some preliminary definitions, our SAT graph-based representation is presented. Formal descriptions of its interesting features are then given. In particular, resolution is notably formulated using graph transitive closure and conditional backbone is also motivated and defined. Finally, using SAT graph-based representation our three proposed applications are then presented. Experimental validation of our proposed approaches are given in Section 5. Interesting paths for future research are given before concluding.

2. Technical preliminaries

Let $\mathcal{B}$ be a Boolean (i.e. propositional) language of formulas built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$) and a set of propositional variables. A *CNF formula* Σ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive or negated propositional variable. For a literal x (respectively a clause c), $\mathcal{V}(x)$ (respectively $\mathcal{V}(c)$) denotes the propositional variable associated to a literal x (respectively the set of variables associated to literals of c). We denote $\mathcal{V}(\Sigma)$ (respectively $\mathcal{L}(\Sigma)$) the set of variables (respectively literals) occurring in Σ. The size of a clause c, denoted $|c|$, is equal to the number of literals it contains. A *unit clause* is a clause of size one. A *unit literal* is the unique literal of a unit clause. *Binary clauses* contains two literals. A *Horn clause* contains at most one positive literal. A clause is called *positive* (respectively *negative*) if all of its literals are positive (respectively negative). A *Horn-SAT* (respectively *2-SAT*) is a formula composed of only Horn (respectively binary) clauses. These two fragments of SAT are known to be solvable in linear time [15,16].

In addition to these usual notations, we define the negation of a set of literals S as the set of the corresponding opposite literals $\bar{S} = \{\neg l \mid l \in S\}$. An *interpretation* I of a Boolean formula ϕ is an assignment of truth values $\{true, false\}$ to its variables. I can be represented as a set (interpreted as a conjunction) of literals with the usual meaning: $\forall p \in \mathcal{V}(\phi)$, if $p \in I$, then $I[p] = true$, otherwise $I[p] = false$. Let $l \in \mathcal{L}(\phi)$, the formula obtained by assigning l to true, is $\phi(l) = \{c - \{\neg l\} \mid c \in \phi, \neg l \in c\} \cup \{c \mid c \in \phi, l \notin c, \neg l \notin c\}$. For an interpretation $I = \{l_1, l_2, \ldots, l_n\}$, $\phi(I) = \phi(l_1)(l_2)\ldots(l_n)$. A *model* of a formula is an interpretation I that satisfies the formula i.e. $\phi(I) = \emptyset$. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist.

Let c_1 be a clause containing a literal a and c_2 a clause containing the opposite literal $\neg a$, one *resolvent* of c_1 and c_2 is the disjunction of all literals of c_1 and c_2 less a and $\neg a$, denoted $res(a, c_1, c_2)$. A resolvent is called *tautological* when it contains opposite literals; otherwise it is called fundamental.

3. SAT-based graph representation

As mentioned in the introduction, different graph representations of CNF formulas have been proposed previously and used for different purposes. Among them, one can mention the implication graph at the basis of many learning schemes [3] and preprocessing techniques [17]. Other kind of graph representation are used for example by survey propagation to solve satisfiable 3-SAT random formulas [18], or for structure visualisation as in [11] and so on. Our proposed representation can be seen as an extension of the 2-SAT graph representation to the general case [15].

3.1. From CNF to labeled graph

Before introducing our approach, let us first recall the well-known graph representation of 2-SAT formulas.

Let Φ be a 2-SAT formula. The graph representation of ϕ, is defined as $G_\phi = (S, E)$, where $S = \{x, \neg x \mid x \in \mathcal{L}(\phi)\}$ and $E = \{(\neg x, y), (\neg y, x) \mid (x \vee y) \in \phi\}$. The polynomial time algorithm usually used to solve ϕ is based on the computation of the transitive closure of the graph G_ϕ (noted $G^c_\phi = (S, E')$). Then, the formula ϕ is unsatisfiable if there exists a literal $x \in S$ such that $(x, \neg x) \in E'$ and $(\neg x, x) \in E'$, otherwise ϕ is satisfiable. Obviously, computing the transitive closure on the graph G_ϕ is equivalent to the saturation of ϕ by applying the resolution rule. Indeed, given two edges (u, v) and (v, w) of E, a new edge (u, w) is generated and added to the graph. The edge (u, w) corresponds to the resolvent $res(v, (\neg u \vee v), (\neg v \vee w)) = (\neg u \vee w)$.

In general, for CNF formulas a natural representation can be obtained using hypergraph where vertices correspond to literals, and hyper-edges to clauses. In the following, a SAT graph based representation of general CNF formulas is presented.

Definition 1. Let c be a clause so that $|c| \geqslant 2$. A *context* η_c associated to c is a conjunction of literals so that $\bar{\eta}_c \subset c$ and $|\eta_c| = |c| - 2$, i.e. when η_c is true the clause c becomes a binary clause.

Example 1. Let $c = (a \vee \neg b \vee c \vee d)$ be a clause. One possible context associated to c is $\eta_c = (b \wedge \neg c)$. The clause c can be rewritten as $((\neg a \wedge \eta_c) \rightarrow d)$.

For a clause c of size k, we have $|\eta_c| = k - 2$. The context associated to a binary clause is empty, whereas for ternary clauses the context is a unit literal. The number of possible contexts of c is equal to $\frac{k(k-1)}{2}$. A clause c of size k can be rewritten in $k(k-1)$ different ways. A unit clause $c = \{a\}$ is represented as $\neg a \rightarrow a$. Using the clause c given in Example 1, we obtain 6 possible contexts of size 2, and 12 different ways for rewriting c.

Let us now define our SAT graph based representation of CNF formulas.

Definition 2 *(Graph SAT representation).* Let ϕ be a CNF formula. We define $G_\phi = (S, E, v)$ the (directed labeled) SAT graph associated to Φ as follows:

- $S = \{x, \neg x \mid x \in \mathcal{L}(\phi)\}$;
- $A = \{a = (\neg x, y) \mid \exists c \in \Phi, c = (x \vee \alpha \vee y)$ where α is a sub-clause of $c\}$. Each edge $a \in A$ is labeled with a set of literals $\bar{\alpha}$ i.e. $v(a) = \bar{\alpha}$.

In the sequel, for clarity reasons, we sometimes denote a labeled edge $a = (x, y)$ as $(x, y, v(a))$. We also denote $cla(a) = (\neg x \vee \overline{v(a)} \vee y)$ as the clause associated to the edge a.

Clearly, Definition 2 generalizes the classical 2-SAT graph representation. Indeed, if all the contexts are empty, i.e. all clauses of ϕ are binary, then all the arcs of G_ϕ are labeled with an empty set of literals.

3.2. Transitive closure/resolution

In this section, a connection between classical resolution and graph transitive closure is described. Let us introduce some necessary definitions. Henceforth, we only consider formula ϕ without tautological clauses, and $G_\phi = (S, A, v)$ the SAT graph representation of ϕ.

Definition 3. Let $G_\phi = (S, A, v)$ be a graph, $a_1 = (x, y, v(a_1)) \in A$ and $a_2 = (y, z, v(a_2)) \in A$. We define $tr(a_1, a_2) = a_3$ such that $a_3 = (x, z, v(a_1) \cup v(a_2) \setminus \{x, \neg z\})$.

In the definition above, the elimination of $\{x, \neg z\}$ from the context associated to a_3 guarantees that the clause $cla(a_3)$ does not contain several occurrences of the same literal. It corresponds to the application of the classical fusion rule, i.e. $(a \vee \alpha \vee a) \equiv (a \vee \alpha)$.

Example 2. Let $a_1 = (x, y, \{\neg z, e\})$ and $a_2 = (y, z, \{x, f\})$. The two clauses encoded in a_1 and a_2 are $c_1 = (\neg x \vee z \vee \neg e \vee y)$ and $c_2 = (\neg y \vee \neg x \vee \neg f \vee z)$ respectively. We obtain $tr(a_1, a_2) = (x, z, \{\neg e, \neg f\})$ and $cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \vee \neg f \vee z)$. This last clause does not contain redundant literals. Using resolution $res(y, c_1, c_2)$ we obtain $c_3 = (\neg x \vee z \vee \neg e \vee \neg x \vee \neg f \vee z)$. Applying fusion rule on c_3 we eliminate one occurrence of $\neg x$ and z. We obtain $res(y, c_1, c_2) = cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \vee \neg f \vee z)$.

Property 1. *Let $c_1 = \{x, y\} \cup \bar{\eta}_{c_1} \in \phi$, $c_2 = \{\neg y, z\} \cup \bar{\eta}_{c_2} \in \phi$, $a_1 = (x, y, \eta_{c_1}) \in A$ and $a_2 = (x, y, \eta_{c_2}) \in A$. We have $res(y, c_1, c_2) = cla(tr(a_1, a_2))$.*

The proof of Property 1 is a direct consequence of Definitions 2 and 3.

Definition 4 *(Path).* A path $p(x, y)$ between x and y in G_ϕ, is defined as follows: $p(x, y) = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$ such that $x_{i_1} = x$ and $x_{i_k} = y$ and $(x_{i_{j-1}}, x_{i_j}) \in A$ for $1 < j \leqslant k$. We define $\eta_p = \bigcup_{1<j\leqslant k} v((x_{i_{j-1}}, x_{i_j}))$ as the context associated to $p(x, y)$ and $tr(p(x, y)) = tr(\ldots(tr((x_{i_1}, x_{i_2}), (x_{i_2}, x_{i_3}))\ldots(x_{i_{k-1}}, x_{i_k}))\ldots)$ as the transitive closure associated to $p(x, y)$.

Definition 5 *(Fundamental path).* Let $p(x, y) = [x = x_{i_1}, x_{i_2}, \ldots, x_{i_k} = y]$ be a path between x and y. $p(x, y)$ is called fundamental if it satisfies the following two conditions:

- η_p does not contain a literal and its opposite;
- $\neg x \notin \eta_p$ and $y \notin \eta_p$.

The following property states that for a fundamental path $p(x, y)$, one can derive a fundamental resolvent using the transitive closure.

Property 2. *Let $p(x, y)$ be a path. If $p(x, y)$ is fundamental iff $cla(tr(p(x, y)))$ is a fundamental clause.*

Proof. For a fundamental path $p(x, y)$, η_p does not contain a literal and its opposite. As $cla(tr(p(x, y))$ can be written as $r = \{\neg x, y\} \cup \bar{\eta}_p$, r is clearly a fundamental clause. Indeed, from Definition 5, η_p does not contain a literal and its opposite (first condition) and $x \in \bar{\eta}_p$ and $\neg y \notin \bar{\eta}_p$ (second condition). The converse is also true. Suppose that $cla(tr(p(x, y)))$ is not fundamental. As $tr(p(x, y)) = (x, y, \eta_p)$. Then $cla(tr(p(x, y))) = \{\neg x, y\} \cup \bar{\eta}_p$ is not fundamental. This means that either the first or the second condition of Definition 5 is violated. □

In the following, we describe how classical resolution can be achieved using graph transitive closure.

Definition 6. Let $G_\phi = (S, A, v)$ a graph representation of ϕ. We define $tr(G_\phi) = (S, A', v')$ as the transitive closure of G_ϕ where $A' = A \cup \{tr(p(x, y)) \mid x \in S, y \in S$ and $p(x, y)$ is a fundamental path$\}$.

Property 3. *Let ϕ be a formula. ϕ is unsatisfiable iff $\exists k$ such that $tr^k(G_\phi) = (S, A', v')$ and $\exists x \in S$ with $(x, \neg x, \emptyset) \in A'$ and $(\neg x, x, \emptyset) \in A'$.*

Proof. Our graph representation is clearly complete, i.e. each clause is encoded in $k(k-1)$ different ways. As shown above, the application of tr on the edges of G_ϕ is equivalent to applying resolution on ϕ. The proof can be derived from the refutation completeness result of classical resolution. □

In Property 3, we have shown how resolution can be performed on the SAT graph representation. Obviously, other SAT techniques (e.g. variable elimination, unit propagation, etc.) can be reformulated using our proposed representation. Let us note that our graph representation admits many interesting features. One of the main advantages is that dependencies between clauses are well expressed using our representation. Such a structural property is known to be important for the efficiency of SAT solvers. Interestingly enough, our representation can be seen as a state graph, where each state (node) represents a literal and the transition between the state s_1 to s_2 can be performed under a given condition $v((s_1, s_2))$ (a set of literals). Many interesting problems can be formulated more easily using graphs. For example, one can be interested in computing the shortest path between a literal and its opposite, i.e. computing the minimal condition for a given literal to be implied.

In the next section, three possible applications of our proposed graph representation are presented.

4. Handling SAT graph representation

In the definition of G_ϕ, each clause is represented $k(k-1)$ times. Such multiple representation of a given clause is only needed for the completeness of the graph transitive closure. That is to say, if all the edges of a given clause are considered, the satisfiability of the formula can be answered from the transitive closure of the graph. In other words, all the possible implications are computed. Let us note that representing each clause with only one edge is sufficient to represent the original formula. Indeed, each edge of the graph corresponds to exactly one clause of the formula.

4.1. 2-SAT strong backdoor sets

In this section, we will show how our graph SAT representation can be used to address the strong backdoor set computation problem. The notion of (strong) backdoor introduced by Williams et al. in [19] is an active research topic because of its connection to problem hardness. A set of variables forms a backdoor for a given formula if there is an assignment to these variables so that the simplified formula can be solved in polynomial time. Such a set of variables is called a strong backdoor if any assignment to these variables leads to a tractable sub-formula. This kind of structure is related to the notion of independent variables [5,20], and to the cycle cutset introduced for constraint satisfaction problem [21].

Let us recall that computing the smallest strong backdoor is an NP-hard problem. In practice, approximating (in polynomial time) a strong backdoor of "reasonable" size is an interesting and important issue. Indeed, the smaller the strong backdoor is, the greater the reduction is in the worst case time complexity. Consequently, the size of the smallest strong backdoor set gives us an upper bound on the worst case complexity.

Previous works have addressed this issue [6]. For example, the authors of [6] try to recover a set of gates (Boolean functions) from a given CNF, then by exploiting variables dependencies a strong backdoor set is computed. The main drawback behind this approach is that many instances do not contain such a particular structure (Boolean functions). Other approaches have been proposed using different techniques such as adapted systematic search algorithm [22]. Recently, in [23] an enhanced concept of sub-optimal reverse Horn fraction of CNF formulas was introduced and an interesting correlation was observed with the satisfiability and the performances of SAT solvers on fixed density random 3-SAT instances. In [24], the relation between the search complexity of unsatisfiable random 3-SAT formulas and the sizes of unsatisfiable cores and strong backdoors were highlighted. More recently, Paris et al. [25] proposed a local search based approach for approximating a strong horn backdoor set of minimal size. The local search algorithm adapted for renaming the CNF formula while maximizing the horn sub formula. The strong horn backdoor set was then computed from the remaining non horn clauses.

As our graph representation is a generalization of the 2-SAT based implication graph, strong backdoor sets are considered with respect to 2-SAT polynomial fragment, i.e. the set B of variables so that for all instantiation I of B the formula ϕ simplified by I belong to 2-SAT.

Let us note that, for computing 2-SAT strong backdoor sets, it is sufficient to consider a restricted SAT graph representation. Each clause of ϕ is represented with only one edge in G_ϕ. We note such restricted graph associated to ϕ, G^{u}_ϕ.

The following property expresses how the SAT graph representation G^{u}_ϕ can be naturally used to compute such a 2-SAT strong backdoor sets.

Property 4. *Let ϕ be a formula, and $G^{\mathrm{u}}_\phi = (S, A, v)$ its associated graph. $B = \bigcup_{a_i \in A}\{\mathcal{V}(l) \mid l \in v(a_i)\}$ is a 2-SAT strong backdoor set.*

Proof. To proove this property it is sufficient to show that for all assignments I of B, $\phi(I)$ is a 2-SAT formula. Each arc $a_i = (x, y, v(a_i)) \in A$ encodes one clause $c = \{\neg x, y\} \cup \overline{v(a_i)} \in \phi$. For each clause $c \in \phi$, all the variables of $v(a_i)$ appears in B. Consequently I either satisfies at least one literal in $\overline{v(a_i)}$ or all the literals in $\overline{v(a_i)}$ are false. In the first case the clause c is satisfied, whereas in the second case c becomes a binary clause. Consequently, B is a 2-SAT strong backdoor. □

Let us recall that our main objective is to approximate the minimal 2-SAT strong backdoor set. To this end, we first use G^{u}_ϕ the unique graph representation of ϕ, i.e. each clause is encoded with only one edge. Secondly, as a clause c of size k can be encoded in several ways, we need to chose one edge among $k(k-1)$ possible ones. To build G^{u}_ϕ, at each step a new clause $c = (x \vee \alpha \vee y) \in \phi$ of size k is processed and a new edge $a = (x, y, v(a))$ where $v(a) = \bar{\alpha}$, is added to the graph.

For each clause c of size k, its associated edge a is chosen as follow: if all the literals of c belongs to the previous computed contexts, the two literals x and y are chosen according to their minimal number of occurrences in ϕ, the other $k-2$ literals are included in $v(a)$. Otherwise, the two literals x and y are chosen from those that do not occur in the previous contexts using the same selection criteria. This selection criteria aims to minimize the number of different literals to be included in the union of the contexts. The reason is that the size of the strong 2-SAT backdoor set clearly depend on the size of the union of the different contexts.

Interestingly enough, the following simple property is used to reduce further the size of the strong backdoor set.

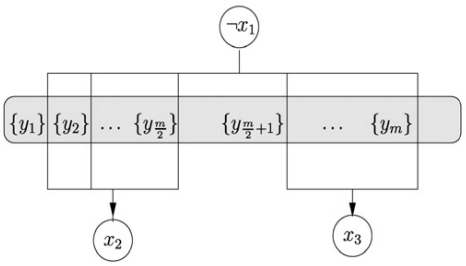
Property 5. *Let B be a 2-SAT strong backdoor set of a CNF formula Φ, and $b \in B$. If $\forall c \in \phi$ so that $b \in c$ or $\neg b \in c$, $|\mathcal{V}(c) \cap B| > |c| - 2$, then b can be eliminated from B.*

The proof of Property 5 is obvious. It expresses the fact that one can eliminate a variable from the 2-SAT strong backdoor B if it appears only in clauses with at most one literal not in B.

Algorithm 1 describes our approach for computing the 2-SAT strong backdoor set. As we consider the 2-SAT fragment, all binary clauses of ϕ are removed (line (2)), then using the heuristic described above, the graph G^{u}_ϕ is built from the remaining set of clauses. A first set B is computed (line (4)), and minimized using Property 5 (lines (5)–(7)).

Algorithm 1. 2-SAT strong backdoor approximation

Input: a cnf formula Φ
Output: a 2-SAT strong backdoor B
begin (1)
  Remove binary clauses from Φ (2)
  Create graph $G^{\mathrm{u}}_\Phi = (S, E)$ (3)
  $B = \bigcup_{a \in E} v(a)$ (4)
  foreach $c \in \Phi$ **do** $nb(c) = |\{x \notin c \cap B\}|$ (5)
  foreach $u \in B$ **do** (6)
    if $(\forall c \in \Phi \mid \{u, \neg u\} \cap c \neq \emptyset)$ **then** $B = B - \{u\}$ (7)
end (8)



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

Fig. 1. A block B_3^m and its associated set of clauses.

4.2. Generating hard SAT instances

In this section, we will show how our graph SAT representation can be used to generate difficult SAT instances. Our construction is based on the graph traversal following paths $p(a, \neg a)$ from a literal a to its opposite $\neg a$. Let us note that when the cumulated set of contexts is empty, then $\neg a$ is an implied literal, i.e. $cla(tr(p(a, \neg a))) = \neg a$. However, deciding if a given literal is implied or finding the minimal condition for its implication is computationally intractable. Our idea consists in constructing a particular graph where the implication of $\neg a$ is easy to prove but difficult to obtain with a given solver. Our proposed generator combine in an original way the graph representation with the well known hyper resolution rule.

Definition 7. Let $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ be a clause, called side clause and ϕ_h be a CNF formula $(\neg y_1 \vee \alpha) \wedge (\neg y_2 \vee \alpha) \wedge \cdots \wedge (\neg y_m \vee \alpha)$, where α is a sub-clause. Applying hyper-resolution on c_s and ϕ_h, in short $hr(c_s, \phi_h)$, we derive the sub-clause α, called hyper-resolvent.

Hyper-resolution can be defined using classical resolution. Indeed, α can also be derived using several resolution steps.

In the following definition, we introduce a new notion, called block-resolution, which extends hyper-resolution. Block-resolution can be seen as an application of several hyper resolution steps.

Definition 8. Let $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ be a side clause and $c_r = (x_1 \vee x_2 \vee \cdots \vee x_k)$ a clause where $1 < k \leqslant m+1$. Let $c_b = c_{h_1} \wedge c_{h_2} \wedge \cdots \wedge c_{h_{k-1}}$ be a CNF formula so that $c_{h_1} = (\neg y_1 \vee x_1 \vee x_2) \wedge \cdots \wedge (\neg y_{\frac{m}{k-1}} \vee x_1 \vee x_2)$ and $c_{h_{k-1}} = (\neg y_{m-\frac{m}{k-1}+1} \vee x_1 \vee x_k) \wedge \cdots \wedge (\neg y_m \vee x_1 \vee x_k)$. We define block resolution on c_s and c_b by $br(c_s, c_b) = hr(\ldots hr(hr(c_s, c_{h_1}), c_{h_2}) \ldots c_{h_{k-1}}) = c_r$, called a block resolvent. We define $B_k^m(x_1, x_2, \ldots, x_k) = c_s \wedge c_b$ as a block formula.

It is important to note, that in Definition 8, for $k = 2$, block-resolution corresponds to the following hyper-resolution rule: $hr(c_s, c_{h_1}) = (x_1 \vee x_2)$ where $c_s = (y_1 \vee y_2 \vee \cdots \vee y_m)$ and $c_{h_1} = (\neg y_1 \vee x_1 \vee x_2) \ldots (\neg y_m \vee x_1 \vee x_2)$.

To illustrate the above definition, in Fig. 1 an example of block formula $B_3^m(x_1, x_2, x_3)$ and its graph representation are given. On the left-hand side of the figure, the side clause is shown in gray box, and the other clauses of $B_3^m(x_1, x_2, x_3)$ are represented using our SAT graph based representation, i.e. each arc from x_1 to x_i with $1 < i \leqslant 3$ is labeled with a different literal from the side clause.

Remark 1. Applying resolution between the side clause and the m other clauses, we can derive the clause $c_r = (x_1 \vee x_2 \vee x_3)$ representing the implication $(\neg x_1 \rightarrow (x_2 \vee x_3))$. Such a resolvent c_r can be obtained with one block-resolution step.

By connecting blocks in a hierarchical form, our goal is to build a graph so as to imply a literal $\neg a$. Our graph is built as shown in Fig. 2. In this figure, box in gray boxes represent side clauses. For clarity reasons, a literal x_i introduced at level k is denoted x_k^i. Starting from the literal a, we use an incremental construction process. Using block-resolution, at each step we generate one more node until a given level l is reached. At this level the generated resolvent obtained by block resolution is of size $l + 1$. In Fig. 2, the obtained resolvent is $(\neg a \vee x_l^1 \vee x_l^2 \vee \cdots \vee x_l^l)$.

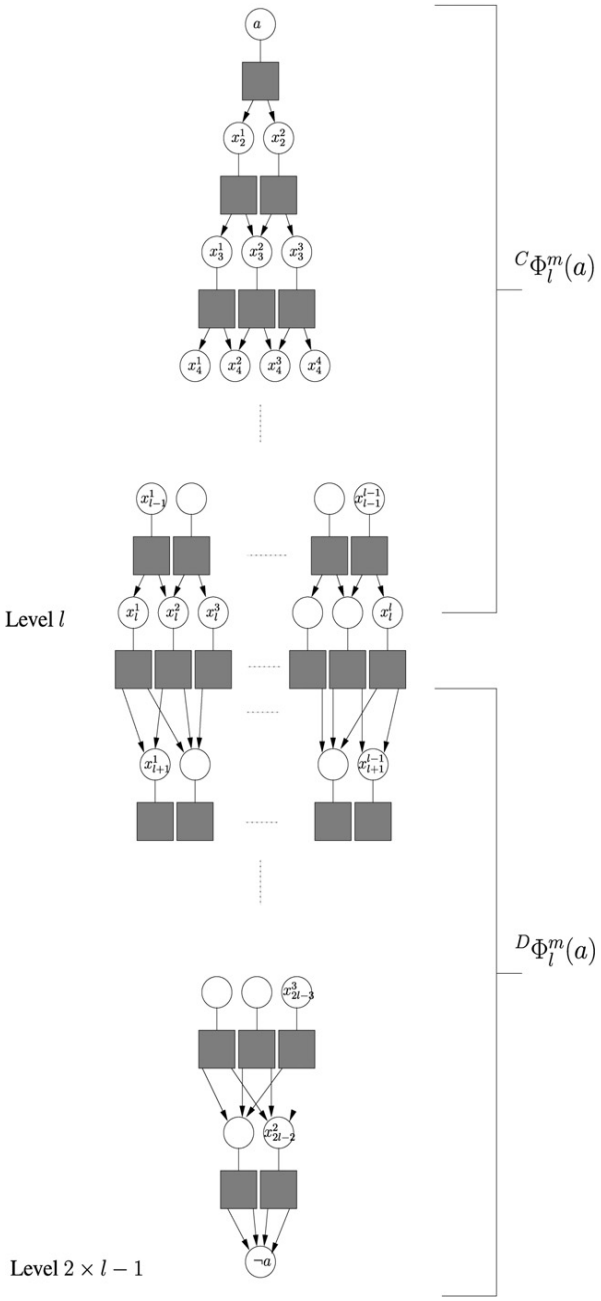


Fig. 2. Graph based generation: a hierarchy of block formulas.

Then, at each step, we remove one node. The last one is the node associated to $\neg a$. At this node, the generated resolvent is $\neg a$. Clearly, from such a graph one derive the literal $\neg a$ using block resolution. However, the number basic resolution step is exponential in l. We formalize this construction in Definition 9.

Definition 9. We define $\Phi_l^m(a) =^C \Phi_l^m(a) \wedge^D \Phi_l^m(a)$ the set of clauses obtained with the partial SAT graph represented in Fig. 2. Each gray box represents a B_3^m block. Φ_l^m is called diamond formula.

The following property ensures that the literal $\neg a$ is implied by the set of clauses $\Phi_l^m(a)$.

Property 6. *Let $\Phi_l^m(a)$ be a diamond formula. We have $\Phi_l^m(a) \models \neg a$.*

Sketch of the proof. Let us recall that for a block $B_3^m(x_1, x_2, x_3)$ can derive $(x_1 \vee x_2 \vee x_3)$ (or $\neg x_1 \rightarrow x_2 \vee x_3$) using block resolution. The traversal of the graph (see Fig. 2) from the top to the bottom level, and applying block-resolution at each step, one can derive successively the following implications: $a \rightarrow x_2^1 \vee x_2^2$, $a \rightarrow x_3^1 \vee x_3^2 \vee x_3^3, \ldots, a \rightarrow x_{2\times l-2}^1 \vee x_{2\times l-2}^2$ and $a \rightarrow \neg a$. □

From this property, it is easy to generate unsatisfiable SAT instances by generating formulas $\Phi_l^m(a) \wedge \Phi_l^m(\neg a)$. However, we observed that these instances are quite easy to solve. To generate hard SAT instances, in our generator, called diamond generator, we consider two different graphs, one for implying a and the other for implying b.

4.2.1. Diamond generator

The generated formulas $\mathcal{G}_l^m$ are a conjunction of two diamond formulas $\Phi_l^m(a) \wedge \Phi_l^m(b)$ where a and b are two different literals. The set of variables of $\mathcal{G}_l^m$ is the union of two disjoint sets of variables $\mathcal{S}$ and $\mathcal{N}$. The set $\mathcal{S}$ represents the set of variables used to build the side clauses (gray boxes in Fig. 2), and $\mathcal{N}$ represents the remaining set of variables (circles in Fig. 2). The formulas $\Phi_l^m(a)$ and $\Phi_l^m(b)$ are generated using the same set of variables $\mathcal{S} \cup \mathcal{N} \setminus \{a, b\}$ and an additional variable a and b, respectively.

The formula $\Phi_l^m(a)$ (respectively $\Phi_l^m(b)$) is generated as in Fig. 2. At each level k,

- All the literals corresponding to side clauses (boxes) and internal nodes (circles) are generated randomly from the set of positive literals associated to $\mathcal{S}$ and from the complete set of literals associated to $\mathcal{N} \setminus \{a, b\}$, respectively.
- The two sets of literals associated to the internal nodes of level k and $k-1$ are disjoint.

As we can see in Section 5 our proposed approach allows to generate small but hard unsatisfiable instances. Furthermore, in our generator only blocks of the form B_3^m (see Definition 9) are considered. Consequently, the generated instances contains only clauses of size 3 and m. Using other kind of blocks, one can derive other instances with clauses of greater size. The proposed generator is available from the authors.

4.3. A graph based preprocessing

Simplifying CNF formulas is an important part of many efficient SAT solvers (e.g. MiniSat [26], Rsat [27]). Many techniques has been proposed over the years. Let us cite "SAT-Elite" [28], one of the most popular and effective technique used as a preprocessing of the state-of-the-art SAT solvers. This technique combines variable elimination and hyper binary resolution [29].

In this section, a new graph based preprocessing of CNF formulas is proposed. As CNF formulas might be very large, a key point behind the design of an efficient preprocessing technique rise in managing the possible overhead with respect to a direct approach, i.e. solving the formula without preprocessing.

In our SAT graph based preprocessing, we consider the following restricted graph representation.

Definition 10 *(Ordered SAT graph).* Let ϕ be a formula. We define the ordered graph SAT representation G_ϕ^o as follows:

- $S = \{x, \neg x \mid x \in \mathcal{L}(\phi)\}$;
- $E = \bigcup_{c\in\phi} e(c)$. For a given $c = (x_1 \vee x_2 \vee \cdots \vee x_k)$, $e(c) = \bigcup_{1\leqslant i\leqslant k} a_i$ where a_i is defined as follows:
  - $(1 \leqslant i < k)$: $a_i = (x_i, x_{i+1}, v(a_i))$ and $v(a_i) = \{x_j \mid 1 \leqslant j \leqslant n$ and $j \neq i$ and $j \neq i+1\}$, and
  - $(i = k)$: $a_k = (x_k, x_1, v(a_k))$ and $v(a_k) = \{x_2, \ldots, x_{k-1}\}$.

Example 3. Let $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x) \wedge (\neg x_2 \vee x) \wedge (\neg x_3 \vee x)$ a formula. The ordered SAT graph representation G_Φ^o is depicted in Fig. 3. As we can remark, on binary clauses, the obtained graph (ordered SAT graph) is the same with and without ordering, i.e. a binary clause is represented by two edges in both graph.

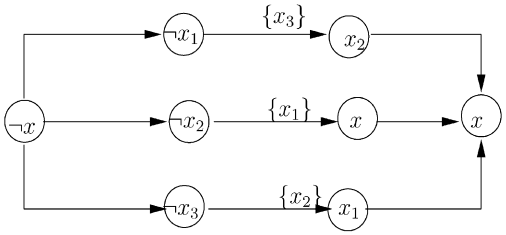


Fig. 3. G^o_Φ of Example 3.

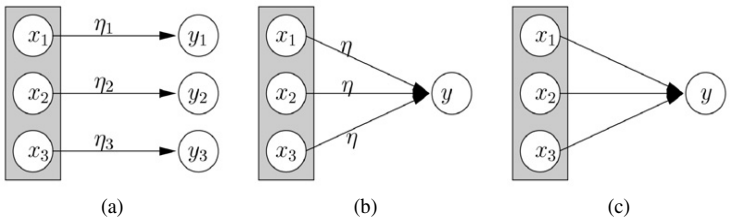


Fig. 4. Extended Hyper Res/Hyper (Bin) Res: a comparison. (a) Extended hyper resolution; (b) Hyper resolution; (c) Hyper binary resolution.

The ordered SAT graph representation gives us a good compromise between a full SAT graph representation and a unique SAT graph one. As we explained in previous sections, using the transitive closure on any path in the graph, one can easily generate new resolvents. We have formulated the resolution method as a graph transitive closure. Consequently, such a process is clearly exponential in the worst case.

Let us now describe the main step of our SAT graph preprocessing. The aim of this step is to generate a fundamental resolvent of bounded length. At each step a new clause $c = (x_1 \vee x_2 \vee \cdots \vee x_n)$ of a given formula ϕ is selected and processed as depicted in Algorithm 2. For each $x_i \in c$ (line (4)), an edge $a = (x_i, x_j, v(a)) \in G^o_\phi$ is chosen so that $res(x_i, c, (\neg x_i \vee \overline{v(a)} \vee x_j))$ is fundamental (lines (4)–(11)). More precisely, the new resolvent is generated following the edges in G^o_ϕ. Such a resolvent is composed of the cumulated set of literals (*nodes*) and contexts (η). This resolvent is obtained using an extension of hyper resolution (see Definition 11). In line (5), an edge a is chosen so that $res(x_i, c, cla(a))$ is fundamental (line (5)). If such edge exists, the context η (respectively *nodes*) is augmented with $a(v)$ (respectively $\{x_j\}$) (line (6)); otherwise x_i is simply added to the set of nodes (line (8)), if the derived resolvent is fundamental (line (7)). In this last case, if no fundamental resolvent can be generated the process is terminated (line (10)).

Definition 11 *(Extended Hyper resolution).* Let $c_s = (x_1 \vee x_2 \vee \cdots \vee x_n)$ be a side clause, and $c_b = \{(\neg x_1 \vee \alpha_1), \ldots, (\neg x_n \vee \alpha_n)\}$ be a set of clauses, where α_i for $1 \leqslant i \leqslant n$ are sub-clauses. The resolvent $\bigcup_{1 \leqslant i \leqslant n} \alpha_i$ as the extended hyper resolvent of c_s and c_b ($ehr(c_s, c_b)$ in short).

Interestingly enough, for formula containing a clause $c = (x_1 \vee x_2 \vee x_3)$ and a set of binary clauses $C = \{(\neg x_1 \vee y), (\neg x_2 \vee y), (\neg x_3 \vee y)\}$, the literal y is generated by Algorithm 2 from the clause c and the edges encoding C. In this case, the application of our algorithm corresponds exactly to the application of hyper binary resolution on c and C [29] (see Fig. 4(b)). For clauses C of arbitrary length ($C = \{(\neg x_1 \vee \neg\eta_1 \vee y_1), (\neg x_2 \vee \neg\eta_2 \vee y_2), (\neg x_3 \vee \neg\eta_3 \vee y_3), \}$) (see Fig. 4(a)), our algorithm generates the resolvent $(\neg\eta_1 \vee \neg\eta_2 \vee \neg\eta_3 \vee y_1 \vee y_2 \vee y_3)$. Clearly, a step of our SAT graph preprocessing can be seen as an extension of the hyper binary resolution proposed in [29] (see Fig. 4(c)).

Our SAT graph preprocessing uses two fixed parameters, the number of considered clauses (*nbCla*) and the maximal size of the generated resolvent (*maxRes*). For each clause c Algorithm 2 is called, then it is iterated using the resolvent generated in the previous step, until reaching the maximal size or no fundamental resolvent can be obtained.

Algorithm 2. Preprocessing step

Input: $G_{\phi}^{or}(V, E)$ and $c \in \Phi$
Output: c_{res} a resolvent clause
begin (1)
  $\eta = \emptyset$ (2)
  $nodes = \emptyset$ (3)
  foreach $x_i \in c$ **do** (4)
    if $\exists a = (x_i, x_j, v(a)) \in E$ so that $x_j \notin \eta$ and $v(a) \cap nodes = \emptyset$ and $\neg x_j \notin c$ and $\neg v(a) \cap c = \emptyset$ (5)
    then $\eta = \eta \cup v(a)$, $nodes = nodes \cup \{x_j\}$ (6)
    elseif ($x_i \notin \eta$ and $\neg x_i \notin nodes$) (7)
    then $nodes = nodes \cup \{x_j\}$ (8)
    else (9)
    break (10)
    $c_{res} = nodes \cup \bar{\eta}$ (11)
end (12)

5. Experiments

In this section, we present the experimental validation of the three proposed applications of our SAT based graph representation.

All the experimental results reported in Section 4 are obtained on a Xeon 3.2 GHz (2 GB RAM). For the strong 2-SAT backdoor approach, the experiments are conducted on instances from the last SAT competitions. The evaluation of our SAT generator is conducted using different SAT solvers. Finally, our preprocessing is validated using the sate-of-the-art SAT solvers on both instances from the SAT competitions and on the instances generated using Diamond generator.

5.1. Strong 2-SAT backdoor set computation

Table 1 exhibits for some instances of the SAT2005 competition the size of the backdoor set obtained by using Algorithm 1 and is compared to the horn strong backdoor computation algorithm proposed in [25]. For each instance, number of variables (#V), number of clauses (#C), size of the 2-SAT strong backdoor computed using our Algorithm 1 and Horn strong backdoor computing using the method proposed in [25] are reported. The time needed to compute the strong backdoor sets is not reported as it never exceed 10 seconds. These results show clearly that our Algorithm 1 computes smallest backdoor set than the one proposed in [25]. In some case, the gain is very impressive (for the `mm` serie for instance). Furthermore, the backdoor size is very short in a lot of case (`equilarge` and `iso` series). Of course, the worst results are obtained on random formulas (`mod2c` serie). So, our graph SAT representation provides an intuitive but powerful method to compute backdoor sets. These results show also that considering the 2-SAT polynomial fragment instead of the horn-SAT one leads to strong backdoor sets of smaller size.

5.2. Evaluating the Diamond generator

In our experiments, different classes of instances $\mathcal{G}_l^m$ are obtained according to different values of m (size of the side clauses) and l (level). Two classes of instances $\mathcal{G}_l^4$ and $\mathcal{G}_l^5$ are considered. For each class, the number of levels (l) vary from 14 to 24. For each fixed m end l, different formulas are then generated by varying the number of variables. For a given number of variables, level and size of the side clauses, 100 instances are generated.

We use three state-of-the-art SAT solvers, ZCHAFF (version 2007.3.12), MINISAT (version 2.0), which are Conflict Driven Clause Learning (CDCL) solvers, and MARCH_EQ (version 010). These three solvers are compared on the generated instances. The CPU time limit is set to 1800 seconds.

Fig. 5 gives the percentage of solved instances by the three solvers for different levels. Fig. 5(a) show the results obtained for $m = 4$. It is clear that the hardness of the generated instances increases with the level l. From level $l = 16$, the generated instances become very hard for the three solvers. At level $l = 23$, the instances are very difficult and none of these solvers is able to solve an instance at level 23. Let us note that, at this level, the generated instances

Table 1
2-SAT vs Horn strong backdoor sets results

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

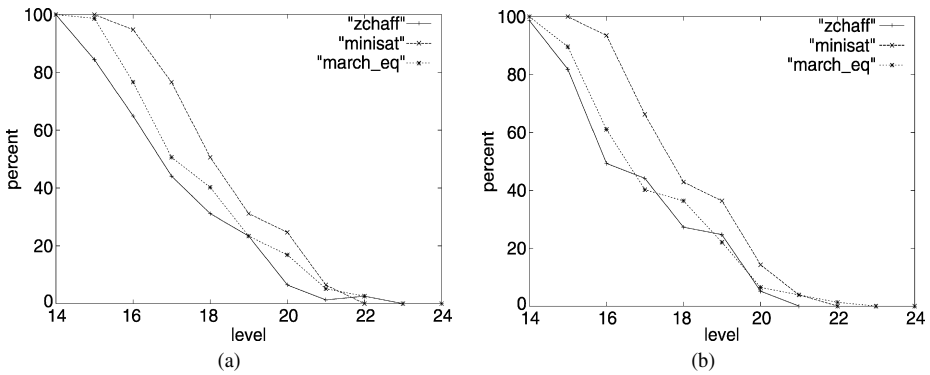


Fig. 5. Percent of instances solved with respect to level. (a) $m = 4$; (b) $m = 5$.

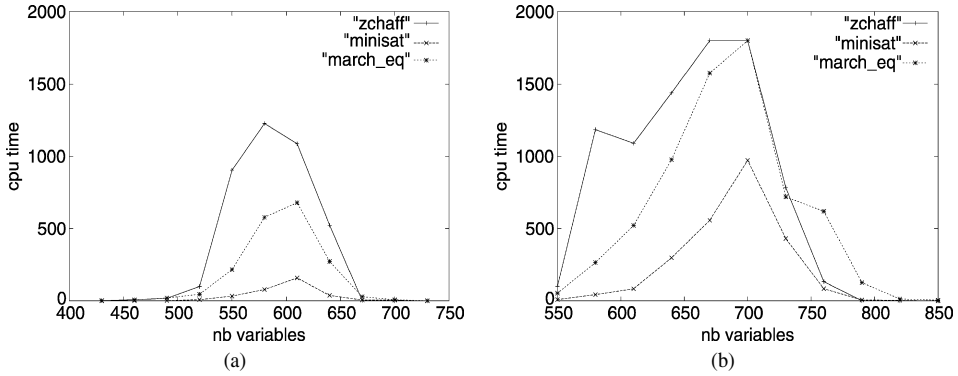


Fig. 6. CPU time comparison ($m = 4$). (a) $l = 15$; (b) $l = 16$.

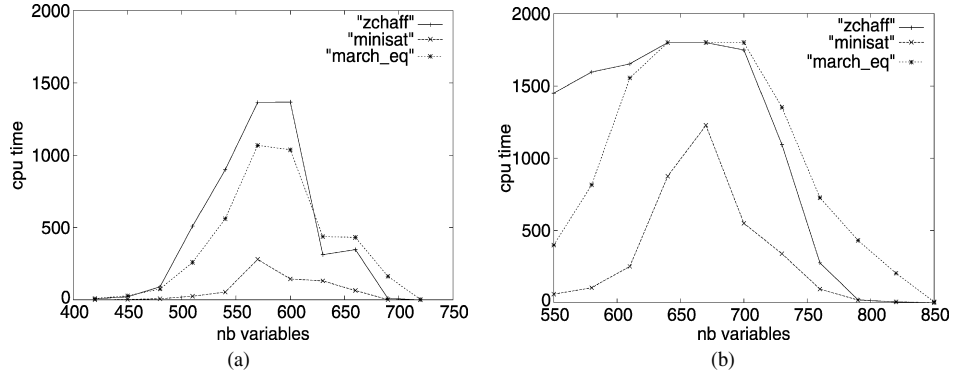


Fig. 7. CPU time comparison ($m = 5$). (a) $l = 15$; (b) $l = 16$.

Table 2
Small and hard SAT instances ($m = 4, 5$)

		ZCHAFF		MINISAT		MARCH_EQ	
Level l	nb inst.	nb solved	avg time	nb solved	avg time	nb solved	avg time
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

contain less than 6000 clauses and the number of variables does not exceed 2000. For $m = 5$ (Fig. 5(b)), generated instances are clearly more difficult than those with $m = 4$. Indeed, at level $l = 20$, ZCHAFF and MARCH_EQ solve 5% of the instances, where MINISAT solves 15%. Here, the number of variables does not exceed 1700.

For a given level, we want to know the impact of the number of variables on the difficulty of the generated instances. As shown in Fig. 6 (where $m = 4$), we observe a threshold phenomenon when the number of variables increases. Indeed, for a given level, it exists a critical value of the number of variables which produce very hard instances. This number is near 600 for a level $l = 15$ (Fig. 6(a)) and near 700 for a level $l = 16$. We can also observe that MINISAT is the best solver on these instances and ZCHAFF obtains the worst results.

This threshold phenomenon is observed also for $\mathcal{G}_l^5$ instances as we can see on Fig. 7. In this case, the critical point is near 580 variables for the level $l = 15$, whereas it is near 680 variables for level $l = 16$.

Tables 2 and 3 show that our generator can produce interesting and difficult satisfiable and unsatisfiable instances. The average time reported in these figures takes only into account cpu-time obtained on solved instances. For a given

Table 3
Small and hard UNSAT instances ($m = 4, 5$)

		ZCHAFF		MINISAT		MARCH_EQ	
Level l	nb inst.	nb solved	avg time	nb solved	avg time	nb solved	avg time
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	–	–

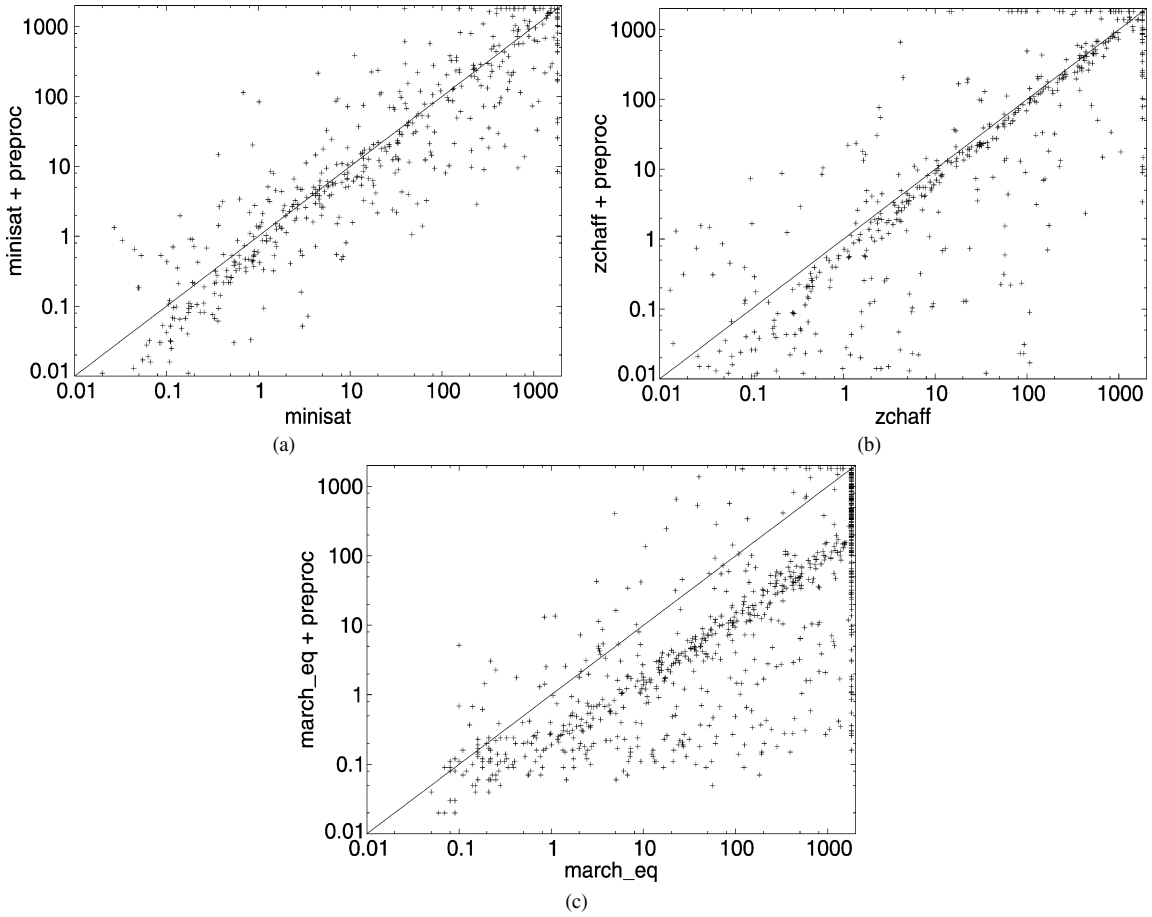


Fig. 8. Results with and without preprocessing. (a) MINISAT; (b) ZCHAFF; (c) MARCH_EQ.

level, we produce difficult satisfiable instances (Table 2) for all solvers. Results reported in Table 3 exhibit small and hard unsatisfiable instances.

The experiments above clearly show that the instances obtained by our generator are very challenging for all the existing SAT solvers.

5.3. Graph based preprocessing

In this section, an experimental evaluation of our graph based preprocessing on both the instances generated using the Diamond generator and on instances from the last SAT competitions is conducted.

First, our preprocessing is evaluated on 2000 instances generated with Diamond generator using different values for the size of the side clauses (m), levels (l), and number of variables (see Section 5.2).

Fig. 8 compares results of the three solvers with and without preprocessing. Each scatter plot given in Fig. 8 illustrates the comparative results of a given solver on each generated instance. The x-axis (respectively y-axis)

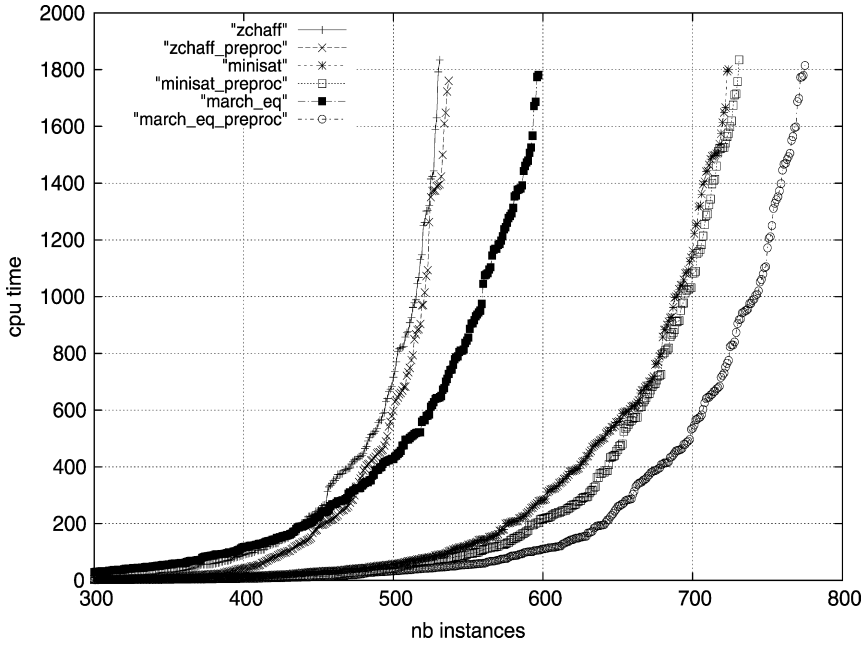


Fig. 9. Number of solved instances vs CPU time.

corresponds to the cpu time tx (respectively ty) obtained by a given solver on the original instance (respectively. instance obtained after applying our preprocessing). Each dot with (tx, ty) coordinates corresponds to a SAT instance. Dots above (respectively below) the diagonal indicate instances where the original formula is solved faster, i.e. $tx < ty$ (respectively slower, i.e. $tx > ty$) than the SAT formula with prepreocessing. This preprocessing step improves significantly the performances of the ZCHAFF and MARCH_EQ. It is more lukewarm for MINISAT.

Finally, the plot in Fig. 9 is obtained as follows: the x-axis represents the number of benchmarks solved by a given solver and the y-axis (in log scale) the time needed to solve this number of instances. In this figure we take into account all generated instances ($\mathcal{G}_l^4$ and $\mathcal{G}_l^5$). It is clear that ZCHAFF and MARCH_EQ are not very good on our benchmarks. MINISAT seems to be the most efficient solver on these instances. Adding the preprocessing step results in a dramatic change of the performances. Each solver is able to solve more instances with these additional clauses generated using our preprocessing (e.g. up to 180 instances for MARCH_EQ). Furthermore, in this case, MARCH_EQ becomes the best solver.

We also conducted the experiments of our sat graph preprocessing proposed in Section 4.3 on other instances taken from the last SAT competitions. Table 4 provides the cpu-time needed for solving some classes of instances by Minisat [4] with and without our preprocessing. To compare our proposed approach with the state-of-the-art preprocessing technique, we also report the results obtained by minisat using SAT-Elite [28]. For each instance, the number of variables (#V), the number of clauses (#C), the number of clauses added to the original formula using our preprocessing (#CA), the cpu time needed to perform our preprocessing (preproc), the total time (preprocessing + solving) needed by Minisat with and without preprocessing are also reported. All times are expressed in seconds. The cpu time limit to solve an instance is fixed to 900 seconds. Since the time to compute resolvent should be very important for huge formulas, we limit to 10% the number of clauses used in the preprocessing step and to 75 the size of clauses added. Here again, the results are very promising and show the potential of the SAT graph representation. Our preprocessing allows to solve 12 instances that are not solved by the classical Minisat solver and good cpu time results are obtained on many classes of instances (e.g. `strips-gripper` and `homer` series). Our preprocessing should also be efficient on large instances (`vmpc` serie). Of course, for easy instances, the preprocessing is useless and generally decreases the original performance of Minisat. As shown in Table 4, our proposed preprocessor obtains interesting results on many instances with respect to SAT-Elite, either on random (`mod`) or structured (`vmpc`) instances.

Table 4
Comparison between Minisat, Minisat + Preprocessing and Minisat + SAT-Elite

Instance	SAT	#V	#C	#CA	preproc	Minisat + preproc	Minisat	Minisat + SAT-Elite
mod2-rand3bip-sat-280-1.as.sat05-2263	Y	280	1120	183	0.76	**574**	–	–
mod2-rand3bip-sat-240-1.as.sat05-2203	Y	240	960	162	0.73	**250**	–	–
mod2-rand3bip-sat-270-1.as.sat05-2248	Y	270	1080	180	0.76	**510**	–	–
mod2-rand3bip-sat-260-2.as.sat05-2234	Y	260	1040	190	0.74	**657**	–	–
mod2c-rand3bip-sat-210-2.as.sat05-2474	Y	297	2092	341	1.14	488	–	**319**
mod2c-rand3bip-sat-180-2.05-2429	Y	252	1784	303	1.08	204	411	**7**
mod2c-rand3bip-sat-170-3.05-2415	Y	240	1724	276	1.04	286	368	**64**
mod2-rand3bip-unsat-150-1.05-2654	N	150	400	79	0.52	**160**	189	784
mod2-rand3bip-unsat-135-2.05-2640	N	135	360	71	0.50	**336**	828.83	–
mod2-rand3bip-unsat-135-1.05-2639	N	135	360	70	0.51	**529**	603	–
mod2-rand3bip-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	96849	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-1200-4800-45-30-003.03-1082	Y	1200	4800	1004	5.82	**85**	197	303
clus-1200-4800-20-20-003.as.sat03-1087	Y	1208	4800	982	6.17	**77**	174	221

6. Conclusion

In this paper we have presented a new graph based representation of Boolean formulas in conjunctive normal form. It extends the well-known graph representation of binary CNF formulas (2-SAT) to the general case. Each clause is represented with different arcs labeled with a set of literals, called contexts. Our proposed representation allows us to extend many interesting features of the 2-SAT polynomial time algorithm. Among them, classical resolution is formulated using the transitive closure of the graph. Furthermore, our approach expresses more naturally the structures of CNF formulas in terms of dependencies and connections between clauses. Such representation of the structures can be exploited for different purposes. In this paper, three possible and promising applications of our SAT graph representation are given, showing its usefulness. The first one deals with the well known strong backdoor set computation problem, whereas in the second the graph representation is used to derive hard SAT instances. Finally, a new preprocessing of SAT instances is proposed. On the first application, we have shown that our representation can be efficiently used to compute the 2-SAT strong backdoor. The obtained results, clearly show that on many instances, the strong backdoor obtained using the 2SAT polynomial fragment clearly outperform the one obtained by Paris et al. [25] using the horn-SAT one. The new proposed preprocessing is clearly promising, many improvements are obtained on both our hard instances and on other instances from the SAT competitions.

The new graph representation of CNF formulas opens many interesting futures works. First, we plan to investigate how such representation can be used to derive new tractable classes of CNF formulas. Secondly, approximating the minimal condition under which a given literal is implied is another interesting perspective. It can be seen as a nice way for circumscribing the backbone literals.

References

[1] B. Selman, H. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, in: Proceedings of Conference on Artificial Intelligence (AAAI), 1994, pp. 440–446.
[2] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Comm. of the ACM 5 (7) (1962) 394–397.
[3] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient sat solver, in: Proceedings of Design Automation Conference, 2001, pp. 530–535.
[4] N. Eén, N. Sörensson, An extensible sat-solver, in: Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), 2003, pp. 502–518.
[5] H. Kautz, D.M.B. Selman, Exploiting variable dependency in local search, in: Abstracts of the Poster Sessions of International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.
[6] E. Gregoire, B. Mazure, R. Ostrowski, L. Sais, Automatic extraction of functional dependencies, in: Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), in: Lecture Notes in Comput. Sci., vol. 3542, 2005, pp. 122–132.
[7] C. Thiffault, F. Bacchus, T. Walsh, Solving non-clausal formulas with DPLL search, in: Proceedings of International Conference on Principles and Practice of Constraint Programming (CP), 2004, pp. 663–678.
[8] H. Dixon, M. Ginsberg, Inference methods for a pseudo-Boolean satisfiability solver, in: Proceedings of the Conference on Artificial Intelligence (AAAI), 2002, pp. 635–640.
[9] F. Aloul, A. Ramani, I. Markov, K.A. Sakallah, Shatter: Efficient breaking for Boolean satisfiability, in: Proceedings of Design Automation Conference, ACM Press, 2003, pp. 836–839.
[10] C. Li, Equivalent literal propagation in the DLL procedure, Discrete Appl. Math. 130 (2) (2003) 251–276.
[11] C. Sinz, E. Dieringer, Dpvis—a tool to visualize the structure of sat instances, in: Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), 2005, pp. 257–268.
[12] A. Darwiche, New advances in compiling CNF to decomposable negational normal form, in: Proceedings of European Conference on Artificial Intelligence, 2004, pp. 328–332.
[13] F. Lu, L. Wang, K. Cheng, R. Huang, A circuit sat solver with signal correlation guided learning, in: Proceedings of Design Automation and Test in Europe, 2003, pp. 892–897.
[14] J. Marques-Silva, K. Sakallah, GRASP: A new search algorithm for satisfiability, in: Proceedings of Computer Aided Design Conference (CAD), 1996, pp. 220–227.
[15] B. Aspvall, M. Plass, R. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Inform. Process. Lett. 8 (3) (1979) 121–123.
[16] W. Dowling, J. Gallier, Linear-time algorithms for testing satisfiability of propositional Horn formulae, J. Logic Program. 3 (1984) 267–284.
[17] R. Brafman, A simplifier for propositional formulas with many binary clauses, in: Proceedings of International Joint Conference on Artificial Intelligence, 2001, pp. 515–522.
[18] A. Braunstein, M. Mézard, R. Zecchina, Survey propagation: An algorithm for satisfiability, Random Structures Algorithms 27 (2) (2005) 201–226.
[19] R. Williams, C. Gomes, B. Selman, Backdoors to typical case complexity, in: Proceedings of International Joint Conference on Artificial Intelligence, 2003, pp. 1173–1178.
[20] E. Giunchiglia, M. Maratea, A. Tacchella, Dependent and independent variables in propositional satisfiability, in: Proceedings of European Conference on Logics in Artificial Intelligence, 2002, pp. 296–307.
[21] R. Dechter, Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition, Artificial Intelligence 41 (3) (1990) 273–312.
[22] P. Kilby, J. Slaney, S. Thiebaux, T. Walsh, Backbones and backdoors in satisfiability, in: Proceedings of the Conference on Artificial Intelligence (AAAI), 2005, pp. 1368–1373.
[23] H.V. Maaren, L.V. Norden, Correlations between horn fractions, satisfiability and solver performance for fixed density random 3-cnf instances, Ann. Math. Artif. Intell. 44 (2005) 157–177.
[24] I. Lynce, J. Marques-Silva, Hidden structure in unsatisfiable random 3-sat: An empirical study, in: Proceedings of International Conference on Tools with Artificial Intelligence, 2004, pp. 246–251.
[25] L. Paris, R. Ostrowski, P. Siegel, L. Sais, Computing horn strong backdoor sets thanks to local search, in: Proceedings of International Conference on Tools with Artificial Intelligence, 2006, pp. 139–143.
[26] N. Eén, N. Sörensson, An extensible sat-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), in: Lecture Notes in Comput. Sci., vol. 2919, Springer, 2003, pp. 502–518.
[27] K. Pipatsrisawat, A. Darwiche, A lightweight component caching scheme for satisfiability solvers, in: Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), 2007, pp. 294–299.
[28] N. Eén, A. Biere, Effective preprocessing in sat through variable and clause elimination, in: Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT), 2005, pp. 61–75.
[29] F. Bacchus, J. Winter, Effective preprocessing with hyper-resolution and equality reduction, in: Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT), 2003, pp. 341–355.

Symmetry breaking in quantified boolean formulae

Gilles AUDEMARD, Said JABBOUR, and Lakhdar SAÏS.

In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2262–2267, 2007.

Symmetry Breaking in Quantified Boolean Formulae

Gilles Audemard* and **Saïd Jabbour** and **Lakhdar Saïs**
CRIL - CNRS, FRE 2499
Université d'Artois - Lens, France
{audemard,jabbour,sais}@cril.univ-artois.fr

Abstract

Many reasoning task and combinatorial problems exhibit symmetries. Exploiting such symmetries has been proved to be very important in reducing search efforts. Breaking symmetries using additional constraints is currently one of the most used approaches. Extending such symmetry breaking techniques to quantified boolean formulae (QBF) is a very challenging task. In this paper, an approach to break symmetries in quantified boolean formulae is proposed. It makes an original use of universally quantified auxiliary variables to generate new symmetry breaking predicates and a new ordering of the QBF prefix is then computed leading to a new equivalent QBF formula with respect to validity. Experimental evaluation of the state-of-the-art QBF solver SEMPROP shows significant improvements (up to several orders of magnitude) on many QBFs instances.

1 Introduction

Solving Quantified Boolean Formulae (QBF) has become an attractive and important research area over the last years. Such increasing interest might be related to different factors, including the fact that many important artificial intelligence (AI) problems (planning, non monotonic reasoning, formal verification, etc.) can be reduced to QBF which is considered as the canonical problem of the PSPACE complexity class. Another important reason comes from the recent impressive progress achieved in the practical resolution of the satisfiability problem. Many solvers for QBFs have been proposed recently (e.g. [Giunchiglia *et al.*, 2001b; Zhang and Malik, 2002; Letz, 2002; Benedetti, 2005]), most of them are obtained by extending satisfiability results. This is not surprising, since QBFs is a natural extension of the satisfiability problem (deciding whether a boolean formula in conjunctive normal form is satisfiable or not), where the variables are universally or existentially quantified.

Some classes of QBFs encoding real-world application and/or AI problems contain many symmetries. Exploiting such structures might lead to dramatically reducing the search space. Symmetries are widely investigated and considered as an important task to deal with the intractability of many combinatorial problems such as constraint satisfaction problems (CSP) and satisfiability of boolean formula (SAT).

A previous work on symmetry breaking predicates for QBF was proposed by [Audemard *et al.*, 2004]. The authors use an hybrid QBF-SAT approach where the set of generated breaking predicates is separated from the QBF formula. Consequently, this approach is solver dependent. Indeed, to solve the hybrid QBF-SAT formula, DPLL-based QBF solvers need to be adapted, whereas, other kind of solvers (e.g. Skizzo) can not be used in this context.

In this paper, we propose a preprocessing approach which is solver independent. Taking as input a QBF with symmetries, we generate a new QBF formula without symmetries equivalent to the original one wrt. validity. To break such symmetries we extend the SAT approach proposed by Crawford [Crawford, 1992; Aloul *et al.*, 2002].

The paper is organized as follows. After some preliminary definitions on quantified boolean formulae, symmetry framework in QBFs is presented. Then a symmetry breaking approach for QBF is described. An experimental validation of our approach is given, showing significant improvements over a wide range of QBF instances. Finally, promising paths for future research are discussed in the conclusion.

2 Technical Background

2.1 Quantified boolean formulae

Let $\mathcal{P}$ be a finite set of propositional variables. Then, $\mathcal{L}_{\mathcal{P}}$ is the language of quantified Boolean formulae built over $\mathcal{P}$ using ordinary boolean formulae (including propositional constants $\top$ and $\bot$) plus the additional quantification ($\exists$ and $\forall$) over propositional variables.

In this paper, we consider quantified boolean formula Φ in the prenex clausal form $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ (in short $QX\Psi$, QX is called a prefix and Ψ a matrix) where $Q_i \in \{\exists, \forall\}$, $X_k, \ldots, X_1$ are disjoint sets of variables and Ψ a boolean formula in conjunctive normal form. Consecutive variables with the same quantifier are grouped. We define $Var(\Phi) = \bigcup_{i\in\{1,\ldots,k\}} X_i$ the set of variables of Φ and $VarU(\Phi) = \{x | x \in Var(\Phi),\ x\ is\ universal\}$. A literal is the occurrence of propositional variable in either positive (l) or negative form ($\neg l$). $Lit(\Phi) = \bigcup_{i\in\{1,\ldots,k\}} Lit(X_i)$ the set

*supported by ANR "Planevo" project n°JC05_41940.

of complete literals of Φ, where $Lit(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$. The set of literals are encoded using integer numbers i.e. the positive (resp. negative) literal l (resp. $\neg l$) is associated to a positive number α (resp. $-\alpha$). Then, we define $|l|$ as the absolute value of l.

For a given variable $x \in Var(\Phi)$ st. $x \in X_k$, we define $rank(x) = k$. Variables appearing in the same quantifier group are equally ranked.

2.2 Symmetries in Quantified Boolean Formulae

Let $\Phi = Q_1 X_1, \ldots, Q_m X_m \Psi$ be a QBF and σ a permutation over the literals of Φ i.e. $\sigma : Lit(\Phi) \mapsto Lit(\Phi)$. The permutation σ on Φ is then defined as follows: $\sigma(\Phi) = Q_1 \sigma(X_1), \ldots, Q_m \sigma(X_m) \sigma(\Psi)$. For example, if Ψ is in clausal form then $\sigma(\Psi) = \{\sigma(c) | c \in \Psi\}$ and $\sigma(c) = \{\sigma(l) | l \in c\}$.

Definition 1 *Let $\Phi = Q_1 X_1, \ldots, Q_m X_m \Psi$ be a quantified boolean formula and σ a permutation over the literals of Φ. σ is a symmetry of Φ iff*

1. $\forall x \in Lit(\Phi), \sigma(\neg x) = \neg \sigma(x)$
2. $\sigma(\Phi) = \Phi$ *i.e* $\sigma(\Psi) = \Psi$ *and* $\forall i \in \{1, \ldots, m\}\ \sigma(X_i) = X_i$.

Let us note that each symmetry σ of a QBF Φ is also a symmetry of the boolean formula Ψ. The converse is not true. So the set of symmetries of Φ is a subset of the set of symmetries of Ψ.

A symmetry σ can be seen as a list of cycles $(c_1 \ldots c_n)$ where each cycle c_i is a list of literals $(l_{i_1} \ldots l_{i_{n_i}})$ st. $\forall 1 \leq k < n_i, \sigma_i(l_{i_k}) = l_{i_{k+1}}$ and $\sigma_i(l_{i_{n_i}}) = l_{i_1}$. We define $|\sigma| = \sum_{c_i \in \sigma} |c_i|$ where $|c_i|$ is the number of literals in c_i.

It is well known that breaking all symmetries might lead in the general case to an exponential number of clauses [Crawford *et al.*, 1996]. In this paper, for efficiency and clarity reasons, we only consider symmetries with binary cycles. Our approach can be extended to symmetries with cycles of arbitrary size.

Detecting symmetries of a boolean formula is equivalent to the graph isomorphism problem [Crawford, 1992; Crawford *et al.*, 1996] (i.e. problem of finding a one to one mapping between two graphs G and H). This problem is not yet proved to be NP-Complete, and no polynomial algorithm is known. In our context, we deal with graph automorphism problem (i.e. finding a one to one mapping between G and G) which is a particular case of graph isomorphism. Many programs have been proposed to compute graph automorphism. Let us mention NAUTY [McKay, 1990], one of the most efficient in practice.

Recently, Aloul *et al.* [Aloul *et al.*, 2002] proposed an interesting technique that transforms CNF formula Ψ into a graph G_Ψ where vertices are labeled with colors. Such colored vertices are considered when searching for automorphism on the graph (i.e. vertices with different colors can not be mapped with each others).

In [Audemard *et al.*, 2004], a simple extension to QBFs formulae is given. Such extension is simply obtained by introducing a different color for each set of vertices whose literals belong to the same quantifier group. In this way, literals from different quantifier groups can not be mapped with each others (see the second condition of the definition 1). Then, to detect such symmetries, NAUTY is applied on the graph representation of the QBF.

3 Breaking symmetries in QBFs

Symmetry breaking has been extensively investigated in the context of constraint satisfaction and satisfiability problems. The different approaches proposed to break symmetries can be conveniently classified as dynamic and static schemes. Dynamic breaking generally search and break symmetries using breaking predicates or not [Benhamou and Sais, 1994; Gent and Smith, 2000]. Static breaking schemes refer to techniques that detect and break symmetries in a preprocessing step. For SAT, symmetries are generally broken by generating additional constraints, called symmetry breaking predicates (SBP) [Crawford, 1992; Aloul *et al.*, 2002]. Such SBP eliminates all models from each equivalence class of symmetric models, except one. However, in the general case, the set of symmetry predicates might be of exponential size. In [Aloul *et al.*, 2002], Aloul *et al* extend the approach of Crawford [Crawford, 1992] by using group theory and the concept of non-redundant generators, leading to a considerable reduction in the SBP size.

We briefly recall the symmetry breaking technique introduced by Crawford in [Crawford *et al.*, 1996]. Let Ψ be a CNF formula and $\sigma = \{(x_1, y_1) \ldots (x_n, y_n)\}$ a symmetry of Ψ. The SBP associated to σ is defined as follows:

$x_1 \leq y_1$
$(x_1 = y_1) \rightarrow x_2 \leq y_2$
$\ldots$
$(x_1 = y_1) \ldots (x_{n-1} = y_{n-1}) \rightarrow x_n \leq y_n$

The SBP defined above expresses that, when for all $i \in \{1 \ldots k-1\}$ x_i and y_i are equivalent (get the same truth value) and x_k is $true$, then y_k must be assigned to $true$. This reasoning can be extended to QBF provided that the symmetry follows the prefix ordering.

3.1 Motivation

In the following example, we show the main difficulty behind the extension of SAT symmetry breaking predicates (SBP) to QBFs.

Example 1 *Let $\Phi = \forall x_1 y_1 \exists x_2 y_2\ \Psi_1$ be a QBF where $\Psi = (x_1 \vee \neg x_2) \wedge (y_1 \vee \neg y_2) \wedge (\neg x_1 \vee \neg y_1 \vee x_2 \vee y_2)$. The permutation $\sigma_1 = \{(x_1, y_1)(x_2, y_2)\}$ is a symmetry of Φ. Breaking the symmetry σ_1 using the traditional approach, induces the following SBP : $(\neg x_1 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_2) \wedge (y_1 \vee \neg x_2 \vee y_2)$. As the clause $(\neg x_1 \vee y_1)$ is universally quantified, the new obtained QBF by adding the SBP to the original QBF leads to an invalid QBF formula.*

To overcome this main drawback, in addition to the classical SBP, new breaking predicates (called $QSBP$) are generated for symmetries containing at least one universal cycle (see definition 3). In such a case, some variables become existentially quantified. These variables will be associated to new additional universally quantified variables. There relationships are expressed in the generated $QSBP$. To safely

add such $QSBP$ to the original QBF formula, a new prefix ordering is computed.

After a formal presentation of our approach for a single symmetry, a generalization to arbitrary set of symmetries is then described.

3.2 Breaking a single symmetry

Now, we formally introduce our approach for breaking symmetries in QBF.

Definition 2 *Let $\Phi = Q_1X_1 \dots Q_iX_i \dots Q_mX_m\ \Psi$ be a QBF and σ a symmetry of Φ. We define $\sigma \uparrow X_i$ as the subsequence of the symmetry σ restricted to the cycles involving variables from X_i. Then, the symmetry σ can be rewriten following the prefix ordering as $\{\sigma_1 \dots \sigma_i \dots \sigma_m\}$ such that $\sigma_i = \sigma \uparrow X_i$. When σ respect the prefix ordering, it is called p-ordered.*

In the sequel, symmetries are considered to be p-ordered.

Example 2 *Let $\Phi = \exists x_2y_2\forall x_1y_1\exists x_3y_3\ (\neg x_1 \vee y_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee y_3)(x_1 \vee x_2 \vee x_3) \wedge (\neg x_3 \vee \neg y_3) \wedge (x_1 \vee x_2)(x_1 \vee y_2) \wedge (\neg x_1 \vee \neg y_1 \vee \neg x_2 \vee \neg y_2)\ \Phi$ has a symmetry $\sigma = \{(x_1, y_1)(x_2, y_2)(x_3, y_3)\}$. Reordering σ with respect to the prefix leads to $\sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ st. $\sigma_1 = \sigma \uparrow X_1 = (x_2, y_2)$, $\sigma_2 = \sigma \uparrow X_2 = (x_1, y_1)$, $\sigma_3 = \sigma \uparrow X_3 = (x_3, y_3)$.*

Definition 3 *Let Φ be a QBF, σ a symmetry of Φ and $c = (x, y)$ is a cycle of σ. We define x (resp. y) as an in-literal (resp. out-literal). A cycle c is called universal if x and y are universally quantified. A symmetry σ is called universal if it contains at least one universal cycle, otherwise it is called existential.*

For existential symmetries of a QBF, classical SBP [Crawford *et al.*, 1996] can be translated linearly to a CNF formula thanks to new additional variables. The obtained set of clauses can be added to the QBF matrix while preserving its validity. The main problem arises when breaking universal symmetries (see example 1). Indeed, to safely break universal symmetries while keeping the classical SBP, we first reorder the symmetry variables belonging to the same universal quantifier group. This new ordering allows us to determine literals to be likely implied from the SBP. Secondly, as implied universal literals lead to the invalidity of the QBF, an original approach is then proposed to deal with such literals. In the following, the problem behind universal implied literals is illustrated and our approach is then motivated.

Let $\sigma = \{(x_1, y_1), \dots, (x_k, y_k), \dots\}$ be a universal symmetry of a given QBF Φ where (x_k, y_k) is a universal cycle. As mentioned above, σ is ordered according to the prefix of Φ. Suppose x_i and y_i for $1 \leq i \leq k-1$ are assigned the same truth value, if x_k is assigned to $true$ then the universal literal y_k is implied from the SBP. To avoid such a case, the universal quantifier of y_k is substituted with an existential quantifier. However, when x_i and y_i are assigned to different truth values or x_k is assigned to false, y_k must remain universally quantified. To manage these two cases, a new universal variable y'_k is then introduced. This variable plays the same role as y_k in the second case whereas in the first case it becomes useless. The relation between the two variables is expressed using new predicates (called $qsbp(\sigma(y_k))$).

Definition 4 *Let $\sigma = \{c_1 \dots c_k \dots c_n\}$ st. $c_k = (x_k, y_k)$, $1 \leq k \leq n$ be a universal symmetry. We define $QSBP(\sigma) = \cup\{qsbp(\sigma(y_k)), 1 \leq k \leq n$ st. y_k is $universal\}$ as the $QSBP$ associated to σ. $QSBP(\sigma)$ is built using the two following steps:*

1. *Adding auxiliary variables : For each universal cycle $c_k = (x_k, y_k) \in \sigma$, we associate a new universal variable y'_k to the out-literal y_k and the universal quantifier of y_k is substituted with an existential quantifier.*
2. *Generating new predicates :*
   - *if (x_1, y_1) is a universal cycle st. $|x_1| \neq |y_1|$ then $qsb(\sigma(y_1)) = \{\neg x_1 \rightarrow (y_1 \leftrightarrow y'_1)\}$*
   - *$\forall k > 1$, if (x_k, y_k) is a universal cycle then $qsbp(\sigma(y_k))$ is made of the following constraints*
     - *$\neg x_k \rightarrow (y_k \leftrightarrow y'_k)$ when $|x_k| \neq |y_k|$*
     - *$((\neg x_j \wedge y_j) \rightarrow (y_k \leftrightarrow y'_k)), \forall j$ st. $1 \leq j < k$*

Example 3 *Let us consider the QBF Φ given in example 1. The symmetry σ of Φ contains one universal cycle (x_1, y_1). Using definition 4, a new variable y'_1 is associated to the variable y_1 which becomes existentially quantified. Then QSBP(σ)= qsbp($\sigma(y_1)$)= $(\neg x_1 \rightarrow (y_1 \leftrightarrow y'_1))$*

As described above, to generate the QSBP new variables are introduced. In the sequel, we describe how such variables are integrated in the QBF prefix.

Definition 5 *Let $\Phi = Q_1X_1 \dots Q_mX_m\ \Psi$ be a QBF, and $\sigma = \{\sigma_1 \dots \sigma_j \dots \sigma_m\}$ be a universal symmetry. Let j st. $Q_j = \forall$ and $\sigma_j = \sigma \uparrow X_j = \{(x_1, y_1) \dots (x_n, y_n)\}$ and $Y' = \{y'_1 \dots y'_n\}$ the set of new variables associated to $\{y_1 \dots y_n\}$ respectively. We define the new ordering of $Var(\sigma_j) \cup Y'$ as follows : $rank(x_k) < rank(y'_k) < rank(y_k)$ st. $1 \leq k \leq n$.*

We define $\mathcal{G}_{\sigma_j}(\mathcal{V}, \mathcal{A})$ as the precedence graph associated to σ_j with $\mathcal{V} = \bigcup_{1 \leq k \leq n}\{x_k, y_k, y'_k\}$ and $\mathcal{A} = \{\bigcup_{1 \leq k \leq n}\{(x_k, y'_k), (y'_k, y_k)\}\}$.

To rewrite the quantifier group Q_jX_j of the QBF formula (see definition 6), a new ordering is derived by applying topological sort algorithm on the precedence graph. Let us note that such a graph is acyclic. In figure 1, the graph representation of $\sigma_j = \sigma \uparrow X_j$ is illustrated. The ordering $x_1 \dots x_n$ $y'_1 \dots y'_n y_1 \dots y_n$ is then considered.

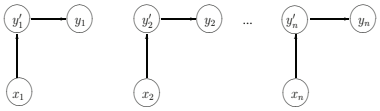


Figure 1: Graph representation of σ_j

Definition 6 *Let $\Phi = Q_1X_1 \dots Q_mX_m\ \Psi$ be a QBF, $\sigma = \{\sigma_1 \dots \sigma_m\}$ a universal symmetry and for $j \in \{1 \dots m\}$ $\sigma_j = \sigma \uparrow X_j = \{(x_1, y_1), \dots (x_n, y_n)\}$. Every quantifier group Q_jX_j of Φ is rewritten :*

- *if $Q_j = \forall$ and $\sigma \uparrow X_j \neq \emptyset$ then*
  $Q^p_jX^p_j = \forall(X_j \backslash Var(\sigma_j))x_1 \dots x_n \exists\alpha\forall y'_1 \dots y'_n \exists y_1 \dots y_n$

(see figure 1) st. $X_j^p = X_j \cup \{y'_1 \dots y'_n\}$ *and* α *is a new variable.*

- *otherwise* $Q_j^p X_j^p = Q_j X_j$

Φ *is then rewritten as*
$\Phi^S = Q_1^p X_1^p \dots Q_i^p X_i^p \dots Q_m^p X_m^p \; \Psi \wedge SBP \wedge QSBP$

Remark 1 *In definition 6, a new variable* α *is introduced to constrain the set of variables* $\{x_1, \dots x_n\}$ *to be assigned before* $\{y'_1, \dots, y'_n\}$.

Property 1 *Let* Φ *be a QBF and* σ *a symmetry of* Φ*, then* Φ *is valid iff* Φ^S *is valid.*

To sketch the proof of property 1, we consider the QBF given in the example 1. Applying our approach on Φ, we obtain the QBF :

$\Phi^s = \forall x_1 \exists \alpha \forall y'_1 \exists y_1 x_2 y_2 \; \Psi_1 \wedge (\neg x_1 \vee y_1) \wedge (\neg x_1 \vee \neg x_2 \vee y_2) \wedge (y_1 \vee \neg x_2 \neg y_2) \wedge (x_1 \vee y_1 \vee \neg y'_1) \wedge (x_1 \vee \neg y_1 \vee y'_1)$
When x_1 is assigned to the value $true$, from the clause $(\neg x_1 \vee y_1)$ we deduce that y_1 is $true$. If x_1 is assigned the value $false$, the original universal variable y_1 is deduced by substitution (y'_1 and y_1 are equivalent) thanks to the added constraint $\neg x_1 \rightarrow (y_1 \leftrightarrow y'_1)$. As $rank(y'_1) < rank(y_1)$, we only need to substitute all occurrences of y_1 by y'_1.

3.3 Breaking all symmetries

Generating QSBP

When considering several symmetries, we can not eliminate them independently by processing each single symmetry using the single symmetry breaking approach described in section 3.2. One needs to consider the interactions between the different symmetries. Indeed, considering an universal out-literal y_k and its associated new variable y'_k, the $qsbp(\sigma(y_k))$ express the conditions under which such literals y_k and y'_k are equivalent. As only one variable y'_k is introduced for each universal out-literal y_k, when y_k appears in several symmetries, the different conditions leading to such equivalence need to be combined.

Definition 7 *Let* Φ *be a QBF, and* $\sigma = \{(x_1, y_1) \dots (x_i, y) \dots\}$. $\sigma' = \{(z_1, w_1) \dots (z_j, y) \dots\}$ *two symmetries of* Φ *with* y *an universal out-literal. The qsbp's associated to* y *with respect to* σ *and* σ' *can be written in the following form (see definition 4) :*

- $qsbp(\sigma(y)) = \{\alpha_1 \rightarrow (y \leftrightarrow y') \dots \alpha_N \rightarrow (y \leftrightarrow y')\}$
- $qsbp(\sigma'(y)) = \{\beta_1 \rightarrow (y \leftrightarrow y') \dots \beta_M \rightarrow (y \leftrightarrow y')\}$.

We define a binary correlation operator η *between* σ *and* σ' *as follows :*
$\eta(\sigma(y), \sigma'(y)) = \{(\alpha_1 \wedge \beta_1) \rightarrow (y \leftrightarrow y') \dots (\alpha_1 \wedge \beta_M) \rightarrow (y \leftrightarrow y') \dots (\alpha_N \wedge \beta_1) \rightarrow (y \leftrightarrow y') \dots (\alpha_N \wedge \beta_M) \rightarrow (y \leftrightarrow y')\}$.

Definition 8 *Let* $\mathcal{S} = \{\sigma^1 \dots \sigma^n\}$ *be a set of symmetries of a given QBF and* y *is universal literal. We define,*

- $\mathcal{S}[y] = \{\sigma^j | \exists (x, y) \in \sigma^j\} = \{\sigma'^1 \dots \sigma'^{|\mathcal{S}[y]|}\}$.
- $qsbp(\mathcal{S}[y]) = \eta(\eta \dots \eta(\sigma'^1, \sigma'^2) \dots, \sigma'^{|\mathcal{S}[y]|}) \dots)$.

Definition 9 *Let* $\mathcal{S} = \{\sigma^1 \dots \sigma^n\}$ *be the set of symmetries of a given QBF* $\Phi = QX \; \Psi$*. The new QBF matrix* Ψ^S *is defined as follows:*

$$\Psi \wedge \Big(\bigwedge_{1 \leq i \leq n} SBP(\sigma^i) \Big) \wedge \Big(\bigwedge_{x \in VarU(\Phi)} qsbp(\mathcal{S}[x]) \Big)$$

Prefix ordering

Let us now show how a new QBF prefix is built when considering a set of symmetries (for a single symmetry see definition 6). For a set of symmetries $\{\sigma^1, \dots, \sigma^n\}$, we consider for each universal quantifier group $Q_k X_k$, all the projections $\sigma^i \uparrow X_k$ for each symmetry σ^i. The new quantifier group $Q_k^p X_k^p$ is obtained from the precedence graph representation of all these projections. Let us recall that each symmetry σ^i is considered p-ordered. Additionally, to avoid cycles from the graph representation of $\{\sigma^i \uparrow X_k | 1 \leq i \leq n\}$, each projection $\sigma^i \uparrow X_k$ is considered lexicographically ordered.

Definition 10 *Let* $\sigma = \{(x_1, y_1), \dots, (x_n, y_n)\}$ *be a set of cycles.* σ *is called lexicographically ordered (lex-ordered in short), iff* $x_i > 0$, $x_i \leq |y_i|$ $1 \leq i \leq n$ *and* $x_i < x_{i+1}$ $1 \leq i < n$

Example 4 *Let* $\Phi = Q_1 X_1 \dots Q_m X_m \; \Psi$ *be a QBF,* σ *and* σ' *two symmetries of* Φ *st.* $\sigma \uparrow X_1 = \{(x_2, x_1)(\neg x_3, \neg x_4)(x_5, x_6)\}$, $\sigma' \uparrow X_1 = \{(x_3, x_1)(x_2, \neg x_6)\}$ *and* $Q_1 = \forall$*. With respect to lexicographical order,* $\sigma \uparrow X_1 = \{(x_1, x_2)(x_3, x_4)(x_5, x_6)\}$ *and* $\sigma' \uparrow X_1 = \{(x_1, x_3)(x_2, \neg x_6)\}$ *Figure 2 shows the precedence graph representation of both* $\sigma \uparrow X_1$ *and* $\sigma' \uparrow X_1$*. Applying topological sort algorithm, the quantifier* $Q_1^P X_1^P$ *is then rewritten as* $Q_1^P X_1^P = \forall x_1 x_5 \exists \alpha \forall x'_2 x'_3 \exists x_2 x_3 \forall x'_4 x'_6 \exists x_4 x_6$

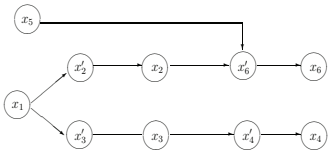


Figure 2: Precedence graph of $\sigma \uparrow X_1$ and $\sigma' \uparrow X_1$

Dealing with universal cycles of the form (y, x) **and** $(z, \neg x)$

To preserve the equivalence (wrt. validity) between the original QBF and the new generated one, let us now address the last problem arising from the interactions between the different symmetries. As illustrated in the following example, the problem arises for symmetries where a universally quantified out-literal x appears both positively (x) and negatively ($\neg x$).

Example 5 *Let* $\Phi = Q_1 X_1 \dots Q_i X_i \dots Q_m X_m \Psi$ *be a QBF,* σ *and* σ' *two symmetries of* Φ *st.* $\sigma \uparrow X_i = \{(x_1, y_2)\}$ *and* $\sigma' \uparrow X_i = \{(x_2, \neg y_2)\}, Q_i = \forall$.

Using our approach the quantifier group $Q_i X_i$ is rewritten as $\forall (X_i \backslash Var(\sigma, \sigma')) \forall x_1 x_2 \exists \alpha \forall y'_2 \exists y_2$. The generated SBP for σ contains the clause $c = (\neg x_1 \vee x_4)$. For σ' its corresponding SBP contains the clause $c' = (\neg x_2 \vee \neg y_2)$. As the universal variable x_4 is substituted with an existential one, applying the $Q\text{-}resolution$ rule [Kleine-Büning *et al.*,

1995] on c and c' leads to a universally quantified resolvent $r = (\neg x_1 \vee \neg x_2)$. Consequently, the new obtained QBF is invalid. Finding a new ordering which avoid this problem is a very challenging task. In example 5, such a problem can be avoided by considering the new ordering $x_1 < y_2 < x_2$ instead of the used lex-ordering. Another possible solution, actually under investigation is to apply composition between symmetries. More precisely, if we consider $\sigma \circ \sigma' \circ \sigma$ we obtain a new symmetry $\sigma'' = (x_1, \neg x_2)$. If in addition to σ and σ', we also consider σ'', then the previous resolvent generated using $Q\text{-}resolution$ is now not universally quantified. Indeed, as x_2 is an out-literal, its universal quantifier is substituted with an existential one. Then the resolvent r contains a literal $\neg x_2$ whose associated variable is existentially quantified. Finally, the quantifier group $Q_i X_i$ can be rewritten as $Q_i^p X_i^p = \forall x_1 \exists \alpha \forall x_2' \exists x_2 \forall y_2' \exists y_2$.

The above discussion gives us an idea on how to solve in the general case, the problem arising from symmetries with universal cycles of the form (y, x) and $(z, \neg x)$. In this paper, such a problem is simply avoided using the following restriction :

Definition 11 *Let $\Phi = Q_1 X_1 \dots Q_m X_m \Psi$ be a QBF, $y \in VarU(\Phi)$ and $\mathcal{S}$ the set of symmetries of Φ. We define $\mathcal{S}[y] \downarrow y = \{\sigma \downarrow y | \sigma \in \mathcal{S}[y]\}$. For $\sigma = \{(x_1, y_1), \dots, (x_k, y_k) \dots, (x_n, y_n)\}$, we define $\sigma\downarrow y_k = \{(x_1, y_1), \dots, (x_{k-1}, y_{k-1})\}$. The restriction of $\mathcal{S}$ wrt. y is defined as $rt(S, y) = \{\sigma | \sigma \in S, \sigma[y] = \emptyset, \sigma[\neg y] = \emptyset\} \cup \mathcal{S}[y] \cup \mathcal{S}[\neg y]\downarrow\{\neg y\}$. For the set of variables $VarU(\Phi) = \{v_1, \dots, v_{|VarU(\Phi)|}\}$, we define $rt(S, VarU(\Phi)) = rt(\dots rt(S, v_1), v_2) \dots v_{|VarU(\Phi)|}) \dots)$*

Note that if $S[y] = \emptyset$ or $S[\neg y] = \emptyset$, then $rt(S, y) = S$. Naturally, for a given set of symmetries S, the new QBF formula is generated using $rt(S, VarU(\Phi))$. In this way the obtained formula is equivalent wrt. validity to the original one.

Complexity

Let σ be a symmetry of a QBF and $CNF(QSBP(\sigma))$ the CNF representation of $QSBP(\sigma)$. The worst case spacial complexity of $CNF(QSBP(\sigma))$ is in $O(|\sigma|^2)$. Considering $\sigma = \{\sigma_1, \dots, \sigma_n\}$ with $\sigma_i = (x_i, y_i)$, $1 \leq i \leq n$, the worst case is reached when all cycles of σ are universal. In this case, $QSBP(\sigma) = \bigcup_{1 \leq i \leq n} qsbp(\sigma_i(y_i))$. $|QSBP(\sigma)| = \sum_{1 \leq i \leq n} |qsbp(\sigma(y_i))|$. $|qsbp(\sigma(y_i))|$ is equal to $2(i-1) + 2 = 2i$ (see definition 4). Then, $|QSBP(\sigma)| = \sum_{1 \leq i \leq n} 2i$ which is equal to $n(n+1)$. More interestingly, using the same new variables introduced for $CNF(SBP)$, the size of $QSBP(\sigma)$ becomes $linear$. Unfortunately, because of the correlations between different symmetries, the $QSBP$ associated to a set of symmetries is exponential in the worst case. In practice, on all the considered QBF instances, the number of applied correlations (definition 7) does not exceed 3. Consequently, the QSBP is most often of reasonable size.

4 Experiments

The experimental results reported in this section are obtained on a Xeon 3.2 GHz (2 GB RAM) and performed on a large panel of symmetric instances (619) available from [Giunchiglia *et al.*, 2001a]. This set of QBF instances contains different families like `toilet`, `k_*`, `FPGA`, `qshifter`. As a comparison, we run the state-of-the-art DPLL-like solver SEMPROP [Letz, 2002] on QBF instances with and without breaking symmetries. The time limit is fixed to 900 seconds. Results are reported in seconds. The symmetry computation time (including detection and QSBP generation) is not reported (less than one second in most cases).

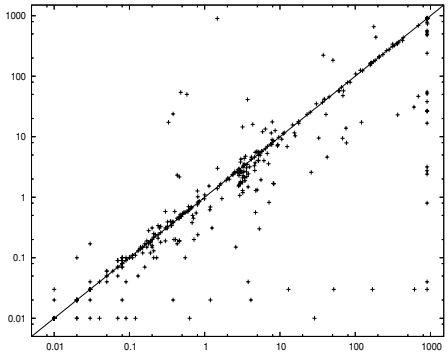


Figure 3: SEMPROP comparison

The scatter plot (in log scale) given in figure 3 illustrate the comprative results of SEMPROP [Letz, 2002] on each QBF instance Φ and Φ^S respectively. The x-axis (resp. y-axis) corresponds to the cpu time tx (resp. ty) obtained by SEMPROP on Φ (resp. Φ^S). Each dot with (tx, ty) coordinates, corresponds to a QBF instance. Dots above (resp. below) the diagonal indicate instances where the original formula Φ is solved faster i.e. $tx < ty$ (resp. slower i.e. $tx > ty$) than the QBF formula Φ^S.

Figure 3 clearly shows the computational gain obtained using symmetry breaking predicates (about 167 instances are solved more efficiently). In some cases the gain is up to 2 orders of magnitude. Of course, there exists some instances where breaking symmetries decreases the performances of SEMPROP(about 50 instances). On the remaining instances, the performance of the solver remains the same with or without breaking symmetries.

Table 1 provides more detailed results on the different QBF families. The second column (NB) represents the number of instances in each family. The third column (U) indicates if the instances contain universal symmetries (Y) or not (N). For each family, S and TT represents the total number of solved instances and the total run-time needed for solving all the instances (900 seconds are added for each unsolved one) respectively.

As we can see, table 1 gives us more comprehensive results with respect to each family. First, breaking symmetries significantly improves SEMPROP performances on many QBF families leading to more solved instances (18 instances). Secondly, the existence of universal symmetries seems to be an important factor for reducing the search time. Not surprisingly, we have also noticed that symmetries between literals occurring in the innermost quantifier group are useless. Indeed, such symmetries does not lead to a great reduction in

			Φ		Φ^S	
family	NB	U	S	TT	S	TT
fpga	8	Y	6	1834	**7**	**921**
blackbox	23	Y	1	19801	**8**	**13668**
scholl	32	Y	17	13687	**18**	**13595**
toilet_c	53	Y	51	2656	**53**	**49**
k_path	40	Y	28	12915	**34**	**7999**
qshifter	6	Y	6	67	6	**61**
tipdiam	76	Y	30	41455	30	41459
asp	104	Y	104	**789**	104	1909
TOILET	7	Y	6	988	6	**926**
term1	6	Y	6	**174**	6	177
strategic	100	N	86	13482	86	**13477**
k_branch	42	N	21	20096	21	**20069**
k_lin	21	N	5	14553	5	14551
k_grz	37	N	23	15242	**24**	**14997**
k_poly	42	N	42	**2031**	42	2068
toilet_a	22	N	22	**12**	22	20
TOTAL	619		454	159789	**472**	**145940**

Table 1: Results on different QBF families

the search tree, since their corresponding variables are assigned last i.e. the formula is considerably reduced by the previous assignments. Finally, for QBF families containing only existential symmetries, breaking them do not improves the search time. On k_branch, k_lin, $toilet\_a$ families containing only existential symmetries, no improvement is observed. Most of these instances correspond to the dots near the diagonal (see figure 3).

5 Conclusion

In this paper, a new approach to break symmetries in QBF formulae is proposed. Using universally quantified auxiliary variables, new symmetry breaking predicates are generated and safely added to the QBF formula. Experimental results show that breaking symmetries leads to significant improvements of the state-of-the-art QBF solver SEMPROP on many classes of QBF instances. These experimental results suggest that for QBF instances containing universal symmetries, significant improvements are obtained. As future works, we plan to investigate the problem arising from the presence of both positive and negative out-literal in the set of symmetries.

References

[Aloul *et al.*, 2002] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. Technical report, University of Michigan, 2002.

[Audemard *et al.*, 2004] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified boolean formulas. In *proceedings of SAT*, 2004.

[Benedetti, 2005] M. Benedetti. sKizzo: a Suite to Evaluate and Certify QBFs. In *Proc. of CADE*, 2005.

[Benhamou and Sais, 1994] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning*, 12(1):89–102, 1994.

[Crawford *et al.*, 1996] J. Crawford, M. Ginsberg, E. Luck, and A. Roy. Symmetry-breaking predicates for search problems. In *proceedings of KR*, pages 148–159. 1996.

[Crawford, 1992] J. Crawford. A theorical analysis of reasoning by symmetry in first order logic. In *Proceedings of Workshop on Tractable Reasonning, AAAI*, pages 17–22, 1992.

[Gent and Smith, 2000] I. Gent and B. Smith. Symmetry breaking in constraint programming. In *Proceedings of ECAI*, pages 599–603, 2000.

[Giunchiglia *et al.*, 2001a] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001.

[Giunchiglia *et al.*, 2001b] E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of IJCAR*, 2001.

[Kleine-Büning *et al.*, 1995] H. Kleine-Büning, M. Karpinski, and A. Flögel. Resolution for quantified boolean formulas. *Information and computation*, 117(1):12–18, 1995.

[Letz, 2002] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux*, pages 160–175, 2002.

[McKay, 1990] B. McKay. nauty user's guide (version 1.5). Technical report, 1990.

[Zhang and Malik, 2002] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the CP*, pages 200–215, 2002.

A symbolic search based approach for quantified boolean formulae

Gilles AUDEMARD and Lakdhar SAÏS.

In F. Bacchus T. Walsh, editor, *proceedings of the eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 3569 of *LNCS*, pages 16–30, 2005.

A Symbolic Search Based Approach for Quantified Boolean Formulas

Gilles Audemard and Lakhdar Saïs*

CRIL CNRS – Université d'Artois,
rue Jean Souvraz SP-18,
F-62307 Lens Cedex France
{audemard, sais}@cril.univ-artois.fr

Abstract. Solving Quantified Boolean Formulas (QBF) has become an important and attractive research area, since several problem classes might be formulated efficiently as QBF instances (e.g. planning, non monotonic reasoning, two-player games, model checking, etc). Many QBF solvers has been proposed, most of them perform decision tree search using the DPLL-like techniques. To set free the variable ordering heuristics that are traditionally constrained by the static order of the QBF quantifiers, a new symbolic search based approach (QBDD(SAT)) is proposed. It makes an original use of binary decision diagram to represent the set of models (or prime implicants) of the boolean formula found using search-based satisfiability solver. Our approach is enhanced with two interesting extensions. First, powerful reduction operators are introduced in order to dynamically reduce the BDD size and to answer the validity of the QBF. Second, useful cuts are achieved on the search tree thanks to the nogoods generated from the BDD representation. Using DPLL-likes (resp. local search) techniques, our approach gives rise to a complete QBDD(DPLL) (resp. incomplete QBDD(LS)) solver. Our preliminary experimental results show that on some classes of instances from the QBF evaluation, QBDD(DPLL) and QBDD(LS) are competitive with state-of-the-art QBF solvers.

Keywords: Quantified boolean formula, Binary decision diagram, Satisfiability.

1 Introduction

Solving quantified boolean formulas has become an attractive and important research area over the last years. Such increasing interest might be related to different reasons including the fact that many important artificial intelligence problems (planning, non monotonic reasoning, formal verification, etc.) can be reduced to QBFs which is considered as the canonical problem of the PSPACE complexity class. Another important reason comes from the recent impressive progress in the practical resolution of the satisfiability problem.

* This work has been supported in part by the IUT de Lens, the CNRS and the Region Nord/Pas-de-Calais under the TAC Programme.

F. Bacchus and T. Walsh (Eds.): SAT 2005, LNCS 3569, pp. 16–30, 2005.

Many solvers for QBFs have been proposed recently (e.g. [11, 18, 12, 10]), most of them are obtained by extending satisfiability results. This is not surprising since QBFs is a natural extension of SAT where the boolean variables are universally or existentially quantified. Most of these solvers take a formula in the prenex clausal form as input and are variant of Davis Logemann and Loveland procedures (DPLL) [7]. However, one of the main drawback of such proposed approaches is that variables are instantiated according to their occurrences in the quantifier prefix (i.e. from the outer to the inner quantifier group). Such preset static ordering limits the efficiency of the search-based QBF solvers. Indeed, in some cases, invalidity of a given QBF might be related to its subparts with variables from the most inner quantifier groups. Consequently, following the order of the prefix might lead to a late and repetitive discovery of the invalidity of the QBF.

The main goal of this paper is to set free the solver from the preset ordering of the QBF and to facilitate the extension of the satisfiability solvers. To this end, binary decision diagrams are used to represent in a compact form the set of models of the boolean formula found by a given satisfiability solver. It give rise to a new QBF solver QBDD(SAT) combining satisfiability search based techniques with binary decision diagram. For completeness and efficiency reasons, our approach is enhanced with two key features. On the one hand, reduction operators are proposed to dynamically reduce at least to some extent the size of the binary decision diagram and to answer the validity of the QBF. On the other hand, for each model found by the satisfiability search procedure, its prime implicant is represented in the BDD and a nogood is returned from the reduced BDD representation and added to the formula. The approach we present in this paper significantly extends our preliminary results [2]. We give a more general framework with additional features such as prime implicants encoding, cuts generation. Two satisfiability search techniques (DPLL and local search techniques) are extended to QBF using our proposed approach.

The paper is organized as follows. After some preliminaries and technical background on quantified boolean formulas and binary decision diagram, it is shown how binary decision diagram can be naturally combined with satisfiability search based techniques to handle QBFs. Using systematic search (respectively stochastic local search) techniques, preliminary experiments on instances of the last QBFs evaluation are presented and show that QBDD(SAT) is competitive and can somtimes achieve significant speedups over state-of-the-art QBF solvers.

2 Preliminaries and Technical Background

Before introducing our approach, we briefly review some necessary definitions and notations about quantified boolean formulas and binary decision diagram.

2.1 Quantified Boolean Formulas

Let $\mathcal{P}$ be a finite set of propositional variables. Then, $\mathcal{L}_{\mathcal{P}}$ is the language of quantified boolean formulas built over $\mathcal{P}$ using ordinary boolean formulas (including propositional constants $\top$ and $\bot$) plus the additional quantification ($\exists$ and $\forall$) over propositional variables.

We consider quantified boolean formula in the prenex form: $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ (in short $QX\Psi$, QX is called the prefix of Φ and Ψ the matrix of Φ) where $Q_i \in \{\exists, \forall\}$, $X_k, \ldots, X_1$ are disjoint sets of variables and Ψ a boolean formula. Consecutive variables with the same quantifier are grouped. The rank of a variable $x \in X_i$ is equal to i (noted $rank(x)$). Variables in the same quantifier group have the same rank value. We define a prefix ordering of QBF formula $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ as the partial ordering obtained according to the decreasing rank of the variables, noted $X_k < X_{k-1} < \cdots < X_1$. A QBF formula Φ is said to be in clausal form if Φ is in prenex form and Ψ is in Conjunctive Normal Form (CNF). Note that we can consider QBFs with inner quantifier Q_1 as existential. Indeed, if Q_1 is a universal quantifier then suppressing $Q_1 X_1$ from the prefix and all occurrences of $x \in X_1$ from the matrix lead to an equivalent QBF. We define $Var(\Phi) = \bigcup_{i \in \{1,\ldots,k\}} X_i$ the set of variables of Φ. A literal is the occurrence of propositional variable in either positive (l) or negative form ($\neg l$). $Lit(\Phi) = \bigcup_{i \in \{1,\ldots,k\}} Lit(X_i)$ the set of complete literals of Φ, where $Lit(X_i) = \{x_i, \neg x_i | x_i \in X_i\}$. We note $var(l)$ the variable associated to a literal l. A literal $l \in \Phi$ is a unit literal iff l is existentially quantified and $\exists c = \{l, l_1, \ldots, l_i\} \in \Psi$ s.t. $\forall l_j, 1 \leq j \leq i$, $var(l_j)$ is universally quantified and $rank(l_j) < rank(l)$. A monotone literal is defined in the usual way as in the pure boolean case (i.e. l is monotone in Φ iff it appears either positively or negatively).

To define the semantic of quantified boolean formulas, let us introduce some necessary notations. Let S be the set of assignments over the set of variables V. The Up-projection (resp. Down-projection) of a set of assignments S on a set of variables $X \subset V$, denoted $S \uparrow X$ (resp. $S \downarrow X$), is obtained by restricting each assignment to literals in X (resp. in $V \backslash X$). The set of all possible assignments over X is denoted by 2^X. An assignment over X is denoted by a vector of literals $\overrightarrow{x}$. In the same way, Up-projection and Down-projection also apply on vector of literals $\overrightarrow{x}$. If $\overrightarrow{y}$ is an assignment over Y s.t. $Y \cap X = \emptyset$, then $\overrightarrow{y}.S$ denotes the set of interpretations obtained by concatenating $\overrightarrow{y}$ with each interpretation of S. Finally, $\Psi(\overrightarrow{x})$ denotes the boolean formula Ψ simplified with the partial assignment $\overrightarrow{x}$. An assignment $\overrightarrow{x}$ is an implicant or a model (resp. nogood) of Ψ; noted $\overrightarrow{x} \models \Psi$ (resp. $\overrightarrow{x} \not\models \Psi$) iff $\Psi(\overrightarrow{x}) = \top$ (resp. $\Psi(\overrightarrow{x}) = \bot$). An implicant (resp. nogood) $\overrightarrow{x}$ is called prime implicant (resp. minimal nogood) of Ψ iff $\nexists \overrightarrow{y} \subset \overrightarrow{x}$ s.t. $\overrightarrow{y} \models \Psi$ (resp. $\overrightarrow{y} \not\models \Psi$).

A QBF formula is valid (is true) if there exists a solution (called a total policy) defined as follows. It is a simplified version of the definition by Sylvie Coste-Marquis *et al.* [13].

Definition 1. *Let $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ a quantified boolean formula and $\pi = \{\overrightarrow{x}_1, \ldots, \overrightarrow{x}_n\}$ a set of models of the boolean formula Ψ. π is a total policy of the quantified boolean formula Φ iff π recursively verifies the following conditions:*

1. *$k = 0$, and $\Psi = \top$*
2. *if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*
3. *if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*

Remark 1. Let π be a total policy of $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$. If $Q_k = \forall$ then we can rewrite π as $\bigcup_{\overrightarrow{x}_k \in 2^{X_k}} \{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$ and if $Q_k = \exists$, then $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ and π can be rewritten as $\{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$

Example 1. Let $\Phi = \exists x_5 x_6 \forall x_2 x_4 \exists x_1 x_3 \Psi$ be a QBF formula, where $\Psi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee x_3) \wedge (x_5 \vee x_6)$. Φ is a valid QBF, since the set of models $\pi = \{(\neg x_5, x_6, x_2, x_4, \neg x_1, x_3), (\neg x_5, x_6, x_2, \neg x_4, \neg x_1, x_3), (\neg x_5, x_6, \neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_5, x_6, \neg x_2, x_4, x_1, x_3)\}$ is a total policy of Φ (Figure 1). The different projection operations are illustrated as follow :

$\pi \uparrow \{x_5, x_6\} = \{(\neg x_5, x_6)\}$

$\pi' = \pi \downarrow \{x_5, x_6\}$

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

$\pi' \uparrow \{x_2, x_4\} = \{(\neg x_2, \neg x_4), (\neg x_2, x_4), (x_2, \neg x_4), (x_2, x_4)\} = 2^{\{x_2, x_4\}}$

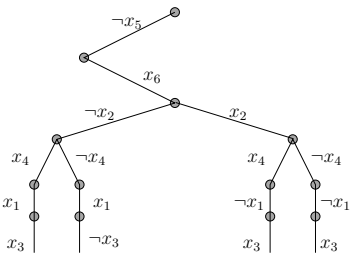


Fig. 1. Policy decision tree representation (example 1)

Motivated by the impressive results obtained in practical solving of the satisfiability problem, several QBF solvers have been developed recently. Most of them are extensions of the well known DPLL procedure including many effective SAT results such as learning, heuristics and constraint propagation (QUBE [11], QUAFFLE [18], EVALUATE [6], DECIDE[16]). For examples, QUBE [11] and QUAFFLE [18] extend backjumping and learning techniques, EVAUATE[6] and DECIDE [16] extend some SAT pruning techniques such as unit propagation.

Algorithm 1 gives a general scheme of a basic DPLL procedure for checking the validity of QBFs. It takes as input variables of the QBF prefix $< X_k, \ldots, X_1 >$ associated to quantifiers $Q_k, \ldots, Q_1$ and a matrix Ψ in clausal form. It returns true if the QBF formula $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ is valid and false otherwise. The algorithm starts by simplifying the formula using unit propagation and monotone literal rules. Then, if the current simplified matrix contains the empty clause then the current QBF is invalid (value false is returned); otherwise, if the current matrix is empty then the QBF formula is valid (value true is returned). The next step consists in choosing the next variable to instantiate (splitting rule) in the most external non empty set of variables X_k. This differ from the DPLL satisfiability version, since variables are instantiated according to their prefix ordering. Depending on the quantifier Q_k of the chosen variable, left and/or right branchs are generated. If $Q_k = \forall$ (resp. $Q_k = \exists$) the right branch is generated

Algorithm 1: *QDPLL* for QBF

Data : Ψ : matrix of the QBF; $< X_k, \ldots, X_1 >$: prefix of the QBF Φ
Result : true if the QBF Φ s valid, false otherwise
begin
Simplify(Ψ);
if $\emptyset \in \Psi$ **then return** false;
if $\Psi = \emptyset$ **then return** true;
if $X_k = \emptyset$ **then return** QDPLL($\Psi, < X_{k-1}, \ldots, X_1 >$);
choose (by heuristic) a literal $l \in X_k$;
if *(($Q_k = \forall$) and QDPLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=false)* **then**
return false;
if *(($Q_k = \exists$) and QDPLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=true)* **then**
return true;
return QDPLL($\Psi \cup \{\neg l\}, < X_k - \{l\}, \ldots, X_1 >$);
end

only if the value returned in left branch is true (resp. false). The search tree developped by the QBF DPLL procedure can be seen as an and/or tree search.

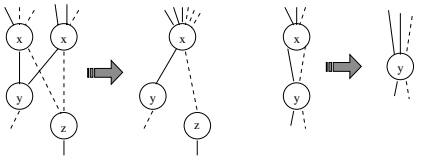
One of the major drawback of the extension of the DPLL procedure to QBF concerns the imposed prefix ordering. Such restrictive ordering might lead to performance degradation of the QBF solver. Since, good ordering might be lost. Furthermore, such limitation makes difficult the extension of certain interesting results obtained on the satisfiability problem (see for example [8] for random problem or [14] for structured problems). One can cite, stochastic local search another search paradigm widely used in SAT (e.g. [17]) that received little attention in QBF. A first integration of local search in QBF solver (WalkQSAT) has been investigated in [10]. WalkQSAT is an implementation of conflict and solution directed backjumping in QBF. It uses a local search solver to guide its search.

2.2 Binary Decision Diagram

A Binary Decision Diagram (BDD) [1, 5] is a rooted directed acyclic graph with two terminal nodes that are referred to as the 0-terminal and the 1-terminal. Every non-terminal node is associated with a primary input variable such that it has two outgoing edges called the 0-edge corresponding to assigning the variable a false truth value, and the 1-edge corresponding to assigning the variable a true truth value. An Ordered Binary Decision Diagram (OBDD) is a BDD such that the input variables appear in a fixed order on all the paths of the graph, and no variable appears more than once in the path. A Reduced Ordered BDD (ROBDD) is an OBDD that results from the repeated application of the following two rules:

1. Share all equivalent sub-graphs (Figure 2.a).
2. Eliminate all redundant nodes whose outgoing edges point to the same node (Figure 2.b).

A (RO)BDD representing a boolean formula Ψ is noted $(RO)BDD(\Psi)$.



a. Sharing sub-formulas b. Redundant node

Fig. 2. Reduction Rules

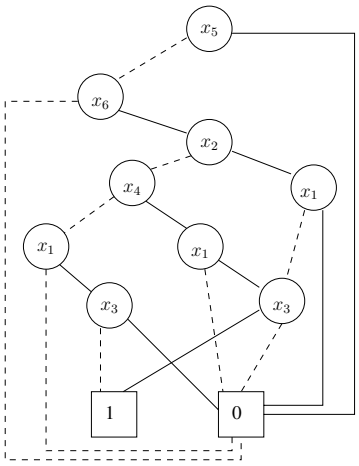


Fig. 3. BDD representation of a policy (example 1)

Figure 3 illustrates the ROBDD representation of the policy π ($ROBDD(\pi)$) shown in example 1 where the solid edges denote the 1-edges and the dashed edges denote the 0-edges. The ROBDD order is the same as the prefix ordering of variables ($\{x_5, x_6\} < \{x_2, x_4\} < \{x_1, x_3\}$) of the QBF Φ.

ROBDDs have some interesting properties. They provide compact and unique representation of boolean functions, and there are efficient algorithms performing all kinds of logical operations on ROBBDs. For example, it is possible to check in constant time whether an ROBDD is true or false. Let us recall that for boolean formula, such problem is NP-complete. Despite the exponential size of the ROBDD in the worst case, ROBDD is one of the most used data structure in practice.

In the rest of this paper, only reduced ordered BDDs are considered and for short we denote them as BDDs. For aquantified boolean formula, the order used in the ROBDD follows the prefix ordering of the QBF.

3 QBDD(SAT): A Symbolic Search Based Approach

In this section, to make the QBF solver freed from the preset ordering of the variables (i.e. fixed by the QBF prefix), we propose an original combination of classical SAT

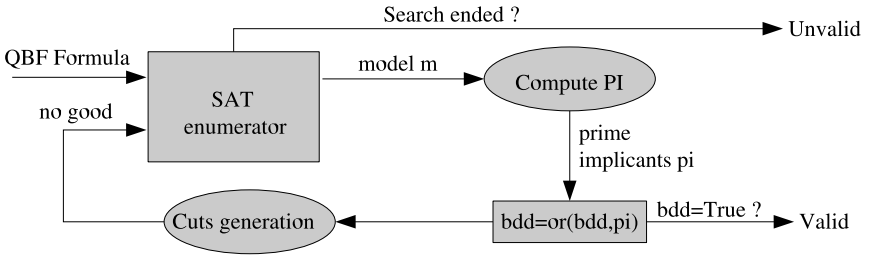


Fig. 4. QBDD(SAT): general scheme

solver with binary decision diagrams. In figure 4, we give a general scheme of our symbolic search based approach QBDD(SAT). More precisely, to check the validity of a QBF $\Phi = QX\Psi$, our approach makes use of a satisfiability technique to search for models (*SAT enumerator*) of the boolean formula Ψ. For each found model m, a prime implicant pi is extracted (*Compute PI*) and disjunctively added to the BDD (*bdd = or(bdd, pi)*) using the prefix ordering of the variables. If the current set of prime implicants represent a total policy then its BDD representation is reduced to 1-terminal node (see section 3.1) and the QBDD(SAT) answer the validity of Φ. As was mentioned earlier, QBDD(SAT) can be instantiated with any satisfiability search technique. For example, QBDD(DPLL) (resp. QBDD(LS)) refer to a QBF solver obtained by instantiating SAT enumerator with DPLL-like (resp. local search) techniques. At the end of the satisfiability search process, if the BDD is not reduced to a 1-terminal node (i.e. a total policy is not found) then depending on the completeness of the SAT used enumerator QBDD(SAT) return either invalid or unknown. Consequently, the QBDD(SAT) is complete iff the satisfiability used solver is also complete.

For space complexity reason, only prime implicants of the boolean formula are encoded in the BDD, nogoods found during the satisfiability search process are not considered. Paths to 0-terminal node in the generated BDD do not represent the nogoods of the boolean formula. Consequently, the 0-terminal node and its incoming edges can be omitted. However, to reduce the search space, for each model (or prime implicant) encoded in the BDD a new nogood is generated and added to the boolean formula (see section 3.2).

3.1 Quantifier Reductions Operators

To reduce the BDD size and to answer the validity of the QBF, additional reduction operator is given in Figure 5. If a node x is existentially quantified and one of its child nodes is the 1-terminal node then any reference to the node x is simply replaced by a reference to its 1-terminal node. We call such reduction operation *existential reduction*. Interestingly enough, when x is universally quantified and its two child nodes are 1-terminal, such node is eliminated using the classical BDD node reduction (Figure 2.b).

During the BDD construction process, in addition to classical reduction operations 2, existential reduction is applied. If the set of models represents a total policy of the QBF formula then the BDD built from such models is reduced to a 1-terminal node as stated by the following property :

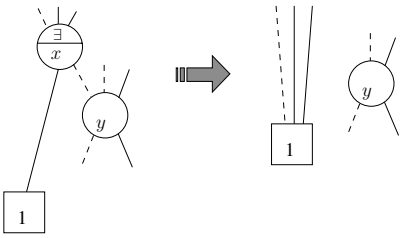


Fig. 5. Existential Reduction

Property 1. Let $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ be a QBF formula and $\pi = \{\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n\}$ a set of models of Ψ. If π is a total policy of Φ then the BDD(π) is reduced to the 1-terminal node.

Proof. The proof is obtained by induction on k. For $k = 0$, the BDD representing the constant $\top$ is a 1-terminal node (by definition of a total policy). Suppose that the property holds for $k-1$, let us prove that it holds for k. By definition of a total policy two case are considered :

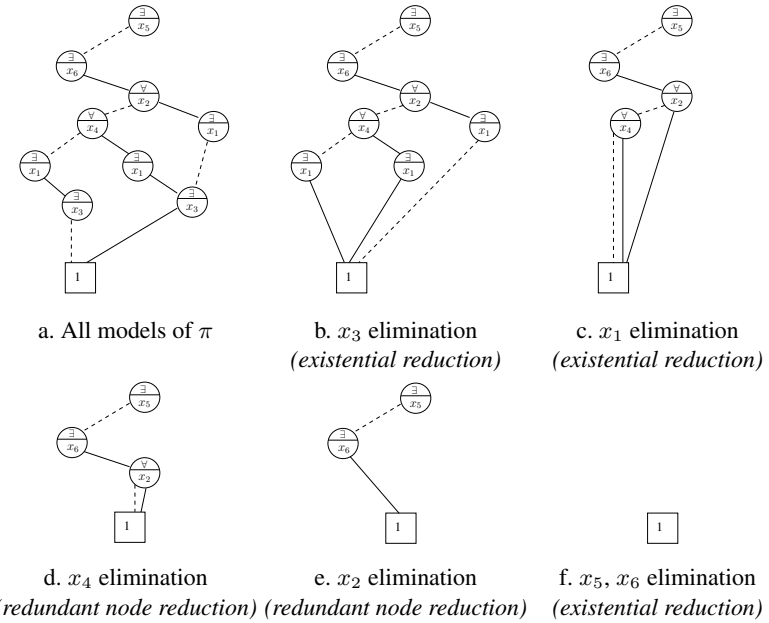
1. if $Q_k = \forall$, then $\pi \uparrow X_k = 2^{X_k}$, and $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1}X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. By induction hypothesis, we can deduce that BDD($\pi \downarrow \overrightarrow{x}_k$) is reduced to 1-terminal node. Consequently all the leaf of the BDD(2^{X_k}) are 1-terminal nodes. Then by repeatedly applying the Redundant node rule on such a BDD, we obtain a BDD reduced to a 1-terminal node.
2. if $Q_k = \exists$, then $\pi \uparrow \overrightarrow{X}_k = \{\overrightarrow{x}_k\}$ and $\pi \downarrow \overrightarrow{x}_k$ is a total policy of the QBF formula $Q_{k-1}X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. By induction hypothesis BDD($\pi \downarrow \overrightarrow{x}_k$) is a 1-terminal node. Consequently, the BDD($\{\overrightarrow{x}_k\}$) can be seen as a branch ended on a 1-terminal node. Repeatedly applying the existential reduction rule, the BDD(π) is reduced to a 1-terminal node.

Remark 2. Dually, *universal reduction* can also be defined. As 0-terminal node of the BDD represents undefined state, then universal reduction operator can not be used in our approach.

The following example shows the dynamic reduction of the BDD associated with the set of models representing the total policy of the QBF given in the example 1.

Example 2. Let Φ be the QBF formula of the example 1 and π its associated policy. The figure 6 represents the reduction phase of the BDD(π) representation:

- The figure 6.a is a BDD representation of the policy π (only paths with final 1-terminal node are represented).
- The existential reduction rule allows the x_3 elimination (figure 6.b) and the x_1 elimination (figure 6.c).
- The redundant node reduction rule allows the x_4 elimination (figure 6.d) and the x_2 elimination (figure 6.e).



a. All models of π | b. x_3 elimination *(existential reduction)* | c. x_1 elimination *(existential reduction)*

d. x_4 elimination *(redundant node reduction)* | e. x_2 elimination *(redundant node reduction)* | f. x_5, x_6 elimination *(existential reduction)*

Fig. 6. BDD reduction of a QBF total policy

– Finally the existential reduction rule suppresses x_5 and x_6 and the bdd of the policy is restricted to the 1-terminal node representing the true formula (figure 6.f) and proving that π is a total policy.

3.2 Generating Cuts from BDD

In order to avoid search of models belonging to different total policies, we introduce in the following different possible cuts (nogoods) that can be generated from the model, prime implicant or from the BDD under construction.

Definition 2. *Let $\Phi = Q_k X_k, \ldots, Q_2 X_2, Q_1 X_1 \Psi$ a QBF s.t. $Q_2 = \forall$, $Q_1 = \exists$ and $\overrightarrow{x}$ is a model of Ψ. We define, $nogood_m(\overrightarrow{x}) = \bigvee\{\neg l | l \in \overrightarrow{x} \downarrow X_1\}$ as the nogood obtained from $\overrightarrow{x}$. In the same way we define $nogood_{pi}(\overrightarrow{pi})$ as the nogood extracted from a prime implicant $\overrightarrow{pi}$.*

Obviously, if $\overrightarrow{pi}$ is a prime implicant obtained from a model $\overrightarrow{x}$ then $nogood_{pi}(\overrightarrow{pi}) \models nogood_m(\overrightarrow{x})$.

Using the example 1, we show in figure 7, that for a model $\overrightarrow{x} = \{\neg x_5, x_6, \neg x_2, x_4, x_1, x_3\}$ the $nogood_m(\overrightarrow{x}) = (x_5 \vee \neg x_6 \vee x_2 \vee \neg x_4)$ ovoid useless search for models of different policies. Considering $\overrightarrow{pi} = \{x_6, x_1, x_3\}$ a prime implicant of $\overrightarrow{x}$, we can generate a strong cut $nogood_{pi}(\overrightarrow{pi}) = \neg x_6$. Interestingly enough, thanks to reductions defined above, the BDD encoding such a prime implicant is reduced to a 1-terminal node and the validity of the QBF is answered.

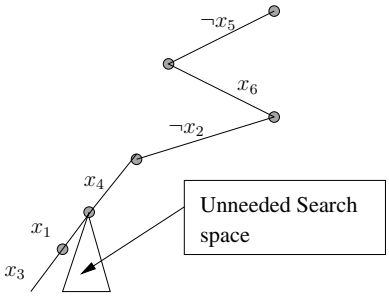


Fig. 7. Cuts generation

Let us recall, that when a prime implicant is disjunctively added to the BDD under construction, the reduction operators eliminate the variables of the inner existential quantifier group X_1. In addition, universally quantified variables in X_2 can be eliminated, when there two outgoing edges point to 1-terminal node. Such a reduction process is iterated recursively. Consequently, to generate strong cuts, in practice each time a prime implicant is added to the BDD, a nogood is extracted from the new reduced BDD and added to the boolean formula.

3.3 QBDD(DPLL) Approach

The algorithm 2 represents the QBF solver obtained by a combination of DPLL procedure with BDD. As we can see, the algorithm search for all models until the BDD representation (characterized by the global variable bdd initialized to the 0-terminal node) is reduced to a 1-terminal node, in that case the algorithm terminates and answers the validity of the QBF formula. If the algorithm backtrack to level 0, then the QBF formula is invalid. In other case search continues (back is returned). The function $Simplify()$ enforces the well known unit propagation process. While the function $conflictAnalysis()$ implements learning scheme used by the most efficient satisfiability solvers. Function $primeImplicants()$ extracts a prime implicant from a model I of the boolean formula Ψ. Computing a prime implicant from a given model can be done in linear time. Indeed, for each literal l of the given model m, we verify if the $m\backslash\{l\}$ is also a model of Psi, in such a case the literal l is deleted from m. Finally, $cutsGeneration()$ generates from the current bdd a new nogood and add it to the cnf Ψ (see section 3.2). Example 3 gives a possible trace of QBDD(DPLL) algorithm.

Example 3. Let us consider the formula Φ of example 1. Suppose that the algorithm QBDD(SAT) starts the search by assigning x_1 and x_5. At this step, the clause $(\neg x_4 \vee x_3)$ is satisfied. If we assign a truth value to the literal $\neg x_4$ then a first model $\{x_5, \neg x_4, x_1\}$ is found and is added to the *bdd*. This variable is reduced to a single path to the 1-terminal node : $< x_5, \neg x_4 >$ (variable x_1 is deleted by existential reduction). The clause $(\neg x_5 \vee x_4)$ is added to the formula Ψ. A backtrack is done to search for other models and the assignment of the literal x_4 implies x_3 using unit propagation. A second model $\{x_5, x_4, x_1, x_3\}$ is added to the `bdd` which becomes reduced to the 1-terminal node using existential and redundant reduction operators. So, search ends and returns the validity of the formula Φ.

Algorithm 2: Combining BDD and DPLL : QBDD(DPLL)

Data : Ψ : set of clauses; X=$\{X_k, \ldots, X_1\}$ the prefix set of the QBF;
I a partial interpretation ; d level in the search tree, initially set to 0
Result : $valid$ if the QBF is valid, $invalid$ otherwise;
$back$ is returned to continue the search for other models.
begin
Simplify(Ψ);
if $\emptyset \in \Psi$ **then**
conflictAnalysis();
return $back$;
if $\Psi = \emptyset$ **then**
$pi := primeImplicants(I, \Psi)$;
bdd := or(bdd, pi);
if *equal(bdd,1-terminal)* **then return** $valid$;
cutsGeneration(pi,bdd);
return $back$;
Let $l \in X_i$ ($i \in \{1 \ldots k\}$) be the chosen branching variable;
if *(*QBDD(DPLL) *($\Psi \cup \{l\}, X - \{l\}, I \cup \{l\}, d+1$)=$valid$) or* QBDD(DPLL)
($\Psi \cup \{\neg l\}, X - \{l\}, I \cup \{\neg l\}, d+1$)=$valid$) **then**
return $valid$;
if *($d = 0$)* **then**
return $invalid$;
return $back$;
end

3.4 QBDD(LS) Approach

One of the important features of our QBDD(SAT) is that any satisfiability search based technique can be used without any major adaptation. Particularly, local search techniques can be integrated in a simple way. Using the state-of-the-art local search WalkSat, we obtain a new incomplete solver QBDD(LS). It answers that the QBF formula is valid, when the set of found models represents a total policy (see property 1); otherwise the solver returns unkown. Our QBDD(LS) solver differs from WalkQSat [10] in the sence that our approach uses local search as a model generator instead of using it to guide the DPLL search procedure.

4 Empirical Evaluation

The experimental results reported in this section are obtained on a Pentium IV 3 GHz with 1GB RAM, and performed on a large panel (644 instances) of the QBF'03 evaluation instances [3]. Theses instances are divided into different families (`log`, `impl`, `toilet`, `k_*`...). For each instance cpu time (in seconds) limit is set to 600 seconds. The QBDD(DPLL) solver is based on chaff like solver called minisat [9], and the QBDD(LS) solver is based on walksat.

4.1 Behaviour of QBDD(DPLL) Solver

Figure 8 presents a pictorial view on the behaviour of different versions of the QBDD (DPLL) solver :

- QBDD(DPLL) is the basic version without computing prime implicants and without generating cuts from the BDD
- QBDD(DPLL) +CUTS is augmented with the generation of cuts from BDD
- QBDD(DPLL) +PI computes prime implicants from a given sat model
- QBDD(DPLL) +PI+CUTS contains these two key features.

The plot in figure 8 is obtained as follows: the x-axis represents the number of benchmarks solved and the y-axis (in log scale) the time needed to solve this number of problems. This figure exhibits clearly that the basic version is the less effective one. Adding only prime implicants does not significantly improve the basic version. However, generating cuts from BDD produces a real improvements, the number of solved problems in less than 600 seconds increases from 130 to 220. Finally, the best version of the QBDD(DPLL) solver is otained by ading cuts and by encoding prime implicants instead of models. Table 1 gives some explanations about these results, it shows the number of models needed to answer the validity of different instances. Since only necessary models are computed with the generation of cuts, the number of models is smaller than without computing cuts. The generation of prime implicants produces some improvements because each prime implicants includes a large potential part of the qbf policies.

Table 2 exhibits the number of instances solved with respect to the size of the quantifier prefix. The method seems to be very efficient with 2QBF formulas and seems to

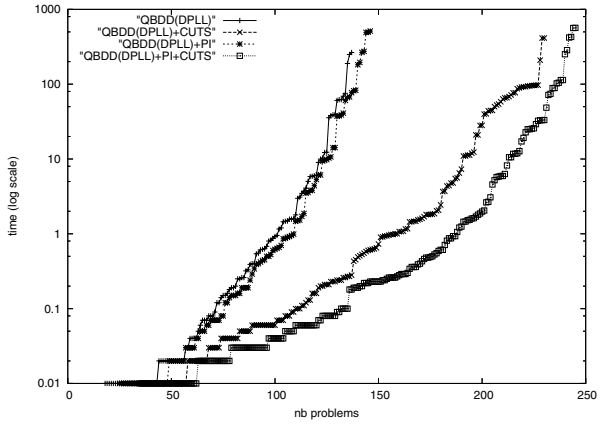


Fig. 8. Number of instances solved vs CPU time

Table 1. Number of models

problem	valid	QBDD(DPLL)	QBDD(DPLL) +PI	QBDD(DPLL) +CUTS	QBDD(DPLL) +PI+CUTS
impl06	Y	6112	6012	2142	550
k_poly_n-1	Y	>45000	>45000	1850	862
toilet_a_10_01.5	N	41412	41412	22486	20 917

Table 2. Solved instances wrt prefix size

prefix size	2	3	4	5	6	7	≥ 8
nb problems	57	339	8	34	7	22	154
nb solved	51	148	1	19	1	11	18
percent	89.5	43.6	12.5	55.8	14.2	50	11.6

be quite ineficcient with large prefix (for example only 11.6% of problems with prefix greater than 8 are solved). However, QBDD(DPLL) is not restricted to 2QBF instances since it solves half of 5-QBF instances. This result agree with those obtained in [15] on 2-QBF using an original combination of two satisfiability solver.

4.2 Comparison with State of the Art Solvers

We compare our solvers QBDD(SAT) to state of the art QBF solvers QUBE [11] and QUANTOR [4]. Here, QBDD(DPLL) solver exploits cuts and prime implicants. Table 3 gives the cpu time comparison between these three solvers on different QBF families instances. *#N* represents the number of problems in the family, *#S* the number of problem solved by a given solver and *TT* the total cpu time needed for a solver to solve all instances of the family. If a method fails to solve an instance then 600 seconds are added to the total cpu time.

Worst results of QBDD(DPLL) are obtained on `toilet` families since only 106 instances are solved in less than 600 seconds, whereas QUBE and QUANTOR solve all quite easily. It's the same for the `k_*` family. Best results are obtained on the `robot` family. Since QBDD(DPLL) solves more and fastly instances than the two other solvers. Furthermore, QBDD(DPLL) solves hard instances of the QBF'03 evaluation for the first time. On a lot of families (`z4`, `flipflop`, `log`) results of all solvers are comparable.

Table 3. CPU time comparison beetwen Qube, Quantor and QBDD(DPLL)

		Qube		Quantor		QBFBDD	
family	#N	#S	TT	#S	TT	#S	TT
robots	48	36	7 214	35	7 970	42	4 270
k_*	171	98	44 813	99	43 786	32	83 658
flipflop	7	7	0.36	7	1.2	7	3.2
toilet	260	260	40	260	256	106	93 860
impl	8	8	0.01	8	0.02	6	1211
tree-exa10	6	6	0.07	6	0.01	6	1.7
chain	8	8	497	8	0.3	2	4183
tree-exa2	6	6	0.01	6	0.01	1	3000.1
carry	2	2	0.23	2	0.69	2	0.71
z4	13	13	0.06	13	0.08	13	0.1
blocks	3	3	0.2	3	0.47	3	3.2
log	2	2	18.4	2	42	2	31

4.3 Comparison with QBDD(LS)

Since QBDD(LS) is an incomplete solver which can not prove the invalidiy of qbf instances, it is quite difficult to make a fair comparison with other qbf solvers. We present its preliminary behaviour on some valid instances and make short comparison with QBDD(DPLL), `Quantor` and `Qube`. Table 4 gives the time to solve an instance and the number of models computed during search (if available). For QBDD(LS), each instance is solved 20 times and the median is reported.

Table 4. Comparison on valid instances

	QBDD(LS)		QBDD(DPLL)		Quantor		Qube	
problem	model	time	model	time	model	time	model	time
impl10	3440	8	2673	2	-	0.01	-	0.01
toilet_c_04_10.2	2850	14	?	>600	-	0.01	-	0.01
robots_1_5_2_3.3	58	6	79	0.28	-	453	-	0.7
comp.blif_0.10_1.00_0_1_out_exact	3205	308	4647	1.8	-	0.03	-	>600
tree_exa10-20	?	>600	1095	0.2	-	0.01	-	0.1

These preliminary experiments shows that QBDD(LS) is quite promising. It solves 76 of the 150 valid instances. It is competitive on some QBF instances and obtain better results than other solvers on some other instances.

5 Conclusion

In this paper, a new symbolic search based approach for QBF is presented. It makes an original combination of model search satisfiability techniques with a binary decision diagrams. BDD are used to encode prime implicants of the boolean formula. Reduction operators that prevent to some extent the blowup of the BDD size and answer the validity of the QBF are presented. Interestingly enough, nogoods are generated from the reduced BDD allowing strong cuts in the SAT search space. The main advantage of our approach is that it is freed from any ordering of the variables. This facilitates the extension of satisfiability solver to deal with quantified boolean formulas. Two satisfiability search paradigms (systematic and local search) have been investigated giving rise to two QBF solvers (QBDD(DPLL) and QBDD(LS)). Experimental results on instances from the QBFs evaluation show the effectiveness of our approach. More interestingly, some open hard QBF instances are solved for the first time.

References

1. S. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27:509–516, 1978.
2. Gilles Audemard and Lakhdar Sais. Sat based bdd solver for quantified boolean formulas. In *proceedings of the 16th IEEE international conference on Tools with Artificial Intelligence*, pages 82–89, 2004.

3. Daniel Le Berre, Laurent Simon, and Armando Tachella. Challenges in the qbf arena: the sat'03 evaluation of qbf solvers. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, *LNAI*, pages 452–467, 2003.
4. Armin Biere. Resolve and expand. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2004.
5. R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–692, C-35.
6. Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, Madison (Wisconsin - USA), 1998.
7. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.
8. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI–01)*, August 4–10 2001.
9. Niklas Eén and Niklas Sörensson. An extensible sat solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, p. 502–508, 2003.
10. Ian P. Gent, Holger H. Hoos, Andrew G. D. Rowley, and Kevin Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proceedings of the 9th international conference of principles and practice of constraint programming*, pages 348–362, 2003.
11. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, Italy, June 2001.
12. Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux 2002*, pages 160–175, Copenhagen, Denmark, 2002.
13. Sylvie Coste Marquis, Helene Fargier, Jerome Lang, Daniel Le Berre, and Pierre Marquis. Function problems for quantified boolean formulas. Technical report, CRIL - France, 2003.
14. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.
15. Darsh Ranjan, Daijue Tang and Sharad Malik Niklas. A Comparative Study of 2QBF Algorithms. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.
16. Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *Proceedings of the First International Conference on Quantified Boolean Formulae (QBF'01)*, pages 84–93, 2001.
17. Bart Selman, Hector Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.
18. Lintao Zhang and Sharad Malik Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 200–215, 2002.

A SAT based approach for solving formulas over boolean and linear mathematical propositions

Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI.

In *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *LNCS*, pages 195–210, 2002.

A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions*

Gilles Audemard[1,2], Piergiorgio Bertoli[1], Alessandro Cimatti[1], Artur Korniłowicz[1,3], and Roberto Sebastiani[1,4]

[1] ITC-IRST, Povo, Trento, Italy
{audemard,bertoli,cimatti,kornilow}@itc.it
[2] LSIS, University of Provence, Marseille, France
[3] Institute of Computer Science, University of Białystok, Poland
[4] DIT, Università di Trento, Povo, Trento, Italy
roberto.sebastiani@dit.unitn.it

Abstract. The availability of decision procedures for combinations of boolean and linear mathematical propositions opens the ability to solve problems arising from real-world domains such as verification of timed systems and planning with resources. In this paper we present a general and efficient approach to the problem, based on two main ingredients. The first is a DPLL-based SAT procedure, for dealing efficiently with the propositional component of the problem. The second is a tight integration, within the DPLL architecture, of a set of mathematical deciders for theories of increasing expressive power. A preliminary experimental evaluation shows the potential of the approach.

1 Introduction

The definition of decision procedures for expressive logical theories, in particular theories combining constraints over boolean and real variables, is a very important and challenging problem. Its importance lies in the fact that problems arising from different real-world domains, ranging from formal verification of infinite state systems to planning with resources, can be easily encoded as decision problems for such theories. The challenge is to define automatic decision procedures, that are able to deal with a wide class of problems, but are also able to recognize easy problems and to deal with them efficiently.

In this paper, we tackle the decision problem for boolean combinations of linear mathematical propositions. We propose an approach based on the extension of efficient DPLL decision procedures for propositional satisfiability with a

* This work is sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102, and has thus benefited of the financial contribution of the Commission through the IHP programme. We thank Andrew Goldberg, Stefano Pallottino and Romeo Rizzi for invaluable suggestions about the problems of solving linear (in)equalities.

A. Voronkov (Ed.): CADE-18, LNAI 2392, pp. 195–210, 2002.

set of mathematical deciders of increasing power. The approach is general and incremental. It allows for the structured integration of mathematical solvers of different expressive power within the DPLL decision procedure, with constraints learning and backjumping. The mathematical solvers have different expressive power, ranging from equalities, to binary linear inequalities, to full linear inequalities. More complex solvers come into play only when needed.

We implemented the approach in the MATH-SATsolver, based on the SIM package for propositional satisfiability. An experimental evaluation was carried out on tests arising from temporal reasoning [2] and formal verification of timed systems [3]. In the first class of problems, we compare our results with the results of the specialized system; although MATH-SAT is able to tackle a wider class of problems, it runs faster that the TSAT solver, that is specialized to a problem class. In the second class, we show the impact of a tighter degree of integration and the different optimization techniques on the ability of the solver. Although preliminary, the experimental evaluation is extremely promising.

The paper is structured as follows. In Section 2 we formalize the class of problems of interest. In Section 3.1 we discuss the general architecture of the solver, and the specific decision procedures that are currently integrated. In Section 4 we present an experimental evaluation, and in Section 5 we describe some related work and draw some conclusions.

2 MATH-SAT

By *math-terms* and *math-formulas* we denote respectively the linear mathematical expressions and formulas built on constants, variables and arithmetical operators over $\mathbb{R}$ and boolean connectives:

- a constant $c_i \in \mathbb{R}$ is a math-term;
- a variable v_i over $\mathbb{R}$ is a math-term;
- $c_i \cdot v_j$ is a math-term, $c_i \in \mathbb{R}$ and v_j being a constant and a variable over $\mathbb{R}$;
- if t_1 and t_2 are math-terms, then $-t_1$ and $(t_1 \otimes t_2)$ are math-terms, $\otimes \in \{+, -\}$.
- a boolean proposition A_i over $\mathbb{B} := \{\bot, \top\}$ is a math-formula;
- if t_1, t_2 are math-terms, then $(t_1 \bowtie t_2)$ is a math-formula, $\bowtie \in \{=, \neq, >, < , \geq, \leq\}$;
- if φ_1, φ_2 are math-formulas, then $\neg\varphi_1$ and $(\varphi_1 \wedge \varphi_2)$ are math-formulas.

The boolean connectives $\vee$, $\rightarrow$, $\leftrightarrow$ are defined from $\wedge$ and $\neg$ in the standard way. For instance, $A_1 \wedge ((v_1 + 5.0) \leq 2.0 \cdot v_3)$ is a math-formula.

An *atom* is any math-formula in one of the forms A_i or $(t_1 \bowtie t_2)$ above —respectively called *boolean atoms* and *mathematical atoms*. A *literal* is either an atom (a *positive* literal) or its negation (a *negative* literal). If l is a negative literal $\neg\psi$, then by "$\neg l$" we conventionally mean ψ rather than $\neg\neg\psi$. We denote by $Atoms(\phi)$ the set of mathematical atoms of a math-formula ϕ.

By *interpretation* is a map $\mathcal{I}$ which assigns real values and boolean values to math-terms and math-formulas respectively and preserves constants, arithmetical and boolean operators:

- $\mathcal{I}(A_i) \in \{\top, \bot\}$, for every $A_i \in \mathcal{A}$;
- $\mathcal{I}(c_i) = c_i$, for every constant $c_i \in \mathbb{R}$;
- $\mathcal{I}(v_i) \in \mathbb{R}$, for every variable v_i over $\mathbb{R}$;
- $\mathcal{I}(t_1 \otimes t_2) = \mathcal{I}(t_1) \otimes \mathcal{I}(t_2)$, for all math-terms t_1, t_2 and $\otimes \in \{+, -, \cdot\}$;
- $\mathcal{I}(t_1 \bowtie t_2) = \mathcal{I}(t_1) \bowtie \mathcal{I}(t_2)$, for all math-terms t_1, t_2 and $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$;
- $\mathcal{I}(\neg\varphi_1) = \neg\mathcal{I}(\varphi_1)$, for every math-formula φ_1;
- $\mathcal{I}(\varphi_1 \wedge \varphi_2) = \mathcal{I}(\varphi_1) \wedge \mathcal{I}(\varphi_2)$, for all math-formulas φ_1, φ_2.

E.g., $\mathcal{I}((v_1 - v_2 \geq 4) \wedge (\neg A_1 \vee (v_1 = v_2)))$ is $(\mathcal{I}(v_1) - \mathcal{I}(v_2) \geq 4) \wedge (\neg\mathcal{I}(A_1) \vee (\mathcal{I}(v_1) = \mathcal{I}(v_2)))$. We say that $\mathcal{I}$ *satisfies* a math formula ϕ, written $\mathcal{I} \models \phi$, iff $\mathcal{I}(\phi)$ evaluates to true. E.g., $A_1 \rightarrow ((v_1 + 2v_2) \leq 4.5)$ is satisfied by an interpretation $\mathcal{I}$ s.t. $\mathcal{I}(A_1) = \top$, $\mathcal{I}(v_1) = 1.1$, and $\mathcal{I}(v_2) = 0.6$.

We call *MATH-SAT* the problem of checking the satisfiability of math-formulas. As standard boolean formulas are a strict subcase of math-formulas, it follows trivially that MATH-SAT is NP-hard.

A *truth assignment* for a math-formula ϕ is a truth value assignment μ to (a subset of) the atoms of ϕ. We represent truth assignments as set of literals

$$\mu = \{\alpha_1, \ldots, \alpha_N, \neg\beta_1, \ldots, \neg\beta_M, A_1, \ldots, A_R, \neg A_{R+1}, \ldots, \neg A_S\}, \qquad (1)$$

$\alpha_1, \ldots, \alpha_N, \beta_1, \ldots, \beta_M$ being mathematical atoms and $A_1, \ldots, A_S$ being boolean atoms, with the intended meaning that positive and negative literals represent atoms assigned to true and to false respectively.

We say that μ *propositionally satisfies* ϕ, written $\mu \models_p \phi$, iff it makes ϕ evaluate to true. We say that an interpretation $\mathcal{I}$ satisfies an assignment μ iff $\mathcal{I}$ satisfies all the elements of μ. For instance, the assignment $\{(v_1 - v_2 \geq 4.0), \neg A_1\}$ propositionally satisfies $(v_1 - v_2 \geq 4.0) \wedge (\neg A_1 \vee (v_1 = v_2))$, and it is satified by $\mathcal{I}$ s.t. $\mathcal{I}(v_1) = 6.0$, $\mathcal{I}(v_2) = 1.0$, $\mathcal{I}(A_1) = \bot$. Intuitively, if we see a math-formula φ as a propositional formulas in its atoms, then $\models_p$ is the standard satisfiability in propositional logic.

Example 1. Consider the following math-formula φ:

$$\begin{aligned}
\varphi = \{&\underline{\neg(2v_2 - v_3 > 2)} \vee A_1\} \wedge \\
&\{\underline{\neg A_2} \vee (2v_1 - 4v_5 > 3)\} \wedge \\
&\{\underline{(3v_1 - 2v_2 \leq 3)} \vee A_2\} \quad \wedge \\
&\{\underline{\neg(2v_3 + v_4 \geq 5)} \vee \underline{\neg(3v_1 - v_3 \leq 6)} \vee \neg A_1\} \quad \wedge \\
&\{A_1 \vee \underline{(3v_1 - 2v_2 \leq 3)}\} \quad \wedge \\
&\{\underline{(v_1 - v_5 \leq 1)} \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \quad \wedge \\
&\{A_1 \vee \underline{(v_3 = 3v_5 + 4)} \vee A_2\}.
\end{aligned}$$

The truth assignment given by the underlined literals above is:

$$\mu = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1), \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4)\}.$$

μ is an assignment which propositionally satisfies φ, as it sets to true one literal of every disjunction in φ. Notice that μ is not satisfiable, as both the following sub-assignments of μ

$$\{(3v_1 - 2v_2 \leq 3), \neg(2v_2 - v_3 > 2), \neg(3v_1 - v_3 \leq 6)\} \quad (2)$$
$$\{(v_1 - v_5 \leq 1), (v_3 = 3v_5 + 4), \neg(3v_1 - v_3 \leq 6)\} \quad (3)$$

do not have any satisfying interpretation. ◇

3 The Solver

3.1 General Idea

The key idea in our approach to solving the MATH-SAT problem consists in stratifying the problem over N layers $L_0, L_1, \ldots, L_{N-1}$ of increasing complexity, and searching for a solution "at a level as simple as possible". In our view, each level considers only an abstraction of the problem which interprets a subgrammar $G_0, G_1, \ldots, G_{N-1}$ of the original problem, G_{N-1} being the grammar G of the problem. Since L_n refines L_{n-1}, if the problem does not admit a solution at level L_n, then it does not at $L_0, \ldots, L_{n-1}$. If indeed a solution S exists at L_n, either n equals $N-1$, in which case S solves the problem, or a refinement of S must be searched at L_{n+1}. In this way, much of the reasoning can be performed at a high level of abstraction. This results in an increased efficiency in the search of the solution, since low-level searches, which are often responsible for most of the complexity, are avoided whenever possible.

The simple and general idea above maps to an N-layered architecture of the solver. In general, a layer L_n is called by layer L_{n-1} to refine a (maybe partial) solution S of the problem. L_n must check for unsatisfability of S and (a) return failure if no refinement can be found, or (b) invoke L_{n+1} upon a refinement S', unless n equals $N-1$. An explanation for failure can be added in case (a), to help higher levels "not to try the same wrong solution twice". L_0 must behave slightly differently, by enumerating (abstract) solutions.

Our solver MATH-SAT realizes the ideas above over the *MATH-SAT* problem. MATH-SAT works on 5 refinement layers. L_0 takes into account only propositional connectives, and is realized by a DPLL propositional satisfiability procedure, modified to act as an enumerator for propositional assignments. To optimize the search, L_0 does not actually ignore mathematical atoms; rather, it abstracts them into boolean atoms, in order to reason upon them at an abstract level. As such, L_0 incorporates an association between newly introduced boolean atoms and originating mathematical atoms, which is used to communicate with L_1. L_1 considers also equalities, performing equality propagation, building equality-driven clusters of variables and detecting equality-driven unsatisfiabilities. L_2 handles also inequalities of the kind $(v_1 - v_2 \bowtie c)$, $\bowtie \in \{<, >, \leq, \geq\}$, by a variant of the Bellman-Ford minimal path algorithm. L_3 considers also general inequalities —except negated equalities— using a standard simplex algorithm. Finally, L_4 considers also negated equalities.

The decomposition in MATH-SAT is significant both because it allows exploiting specialized efficient algorithms to deal with each layer, and because a number of significant problems can be expressed using one of the subgrammars

```
boolean MATH-SAT(formula φ, interpretation & I)
    μ = ∅;
    return MATH-DPLL(φ, μ, I);

boolean MATH-DPLL(formula φ, assignment & μ, interpretation & I)
    if (φ == ⊤) {                                   /* base        */
        I = MATH-SOLVE(μ) ;
        return (I ≠ Null) ; }
    if (φ == ⊥)                                     /* backtrack */
        return False;
    if {a literal l occurs in φ as a unit clause}   /* unit propagation   */
        return MATH-DPLL(assign(l, φ), μ ∪ {l}, I);
    l = choose-literal(φ);                          /* split       */
    return   (MATH-DPLL(assign(l, φ), μ ∪ {l}, I)   or
              MATH-DPLL(assign(¬l, φ), μ ∪ {¬l}, I) );
```

Fig. 1. Pseudo-code of the basic version of the MATH-SAT procedure.

G_0, G_1, G_2. For instance, classical planning problems can be encoded in G_0, both the solving of disjunctive temporal constraints and the reachability of timed systems can be encoded in G_2. In those cases, the specialized search algorithms are used, so efficiency is not sacrificed to expressivity.

3.2 L_0: The Boolean Solver

To solve the satisfiability problem for our math-formulas, we have implemented a solver based on a variant of DPLL, along the guidelines described in [16]. The basic schema of such a procedure, called MATH-SAT, is reported in Figure 1. MATH-SAT is sound and complete [16].

MATH-SAT takes as input a math-formula φ and returns a truth value asserting whether φ is satisfiable or not, and in the former case an interpretation $\mathcal{I}$ satisfying φ. MATH-SAT is a wrapper for the main routine, MATH-DPLL. MATH-DPLL looks for a truth assignment μ propositionally satisfying φ which is satisfiable from the mathematical viewpoint. This is done recursively, according to the following steps:

- (base) If $\varphi = \top$, then μ propositionally satisfies φ. Thus, if μ is satisfiable, then φ is satisfiable. Therefore MATH-DPLL invokes MATH-SOLVE(μ), which returns an interpretation for μ if it is satisfiable, $Null$ otherwise. MATH-DPLL returns $True$ in the first case, $False$ otherwise.
- (backtrack) If $\varphi = \bot$, then μ has lead to a propositional contradiction. Therefore MATH-DPLL returns $False$.
- (unit) If a literal l occurs in φ as a unit clause, then l must be assigned $\top$.Thus, MATH-DPLL is recursively invoked upon *assign*(l, φ) and the assignment obtained by adding l to μ. *assign*(l, φ) substitutes every occurrence of l in φ with $\top$ and propositionally simplifies the result.

– (split) If none of the above situations occurs, then *choose-literal(φ)* returns an unassigned literal l according to some heuristic criterion. Then MATH-DPLL is first invoked upon *assign(l, φ)* and $\mu \cup \{l\}$. If the result is $False$, then MATH-DPLL is invoked upon *assign($\neg l, \varphi$)* and $\mu \cup \{\neg l\}$.

MATH-DPLL is a variant of DPLL, modified to work as an enumerator of truth assignments, whose satisfiability is recursively checked by MATH-SOLVE. The key difference wrt. standard DPLL is in the "base" step. Standard DPLL needs finding only one satisfying assignment μ, and thus simply returns $True$. MATH-DPLL instead also needs checking the satisfiability of μ, and thus it invokes MATH-SOLVE(μ). Then it returns $True$ if a non-null interpretation satisfying μ is found, it returns $False$ and backtracks otherwise.

The search space of the MATH-SAT problem for a math-formula φ is *infinite*. However, MATH-DPLL partitions such space into a *finite* number of regions, each induced by the mathematical constraints in one assignment μ propositionally satisfying φ. Each such region may contain an up-to-infinite set of satisfying interpretations. If so, MATH-SOLVE picks and returns one of them. Also, since MATH-SOLVE works in polynomial space, MATH-SAT works in polynomial space.

3.3 L_1-L_4: The Mathematical Solver

MATH-SOLVE takes as input an assignment μ, and returns either an interpretation $\mathcal{I}$ satisfying μ or $Null$ if there is none. (For simplicity we assume to rewrite all the negated mathematical literals in μ into positive atoms, e.g., $\neg(t_1 = t_2) \Longrightarrow (t_1 \neq t_2)$, $\neg(t_1 > t_2) \Longrightarrow (t_1 \leq t_2)$, etc.)

L_1: Eliminating equalities. The first step eliminates from μ all equalities and simplifies μ accordingly. First, all atoms in the form $(v_i = v_j)$ are removed from μ and all variables occurring there are collected into equivalence classes $E_1, ..., E_i, ..., E_k$, and for each E_i a representant variable $v_i' \in E_i$ is designated. Then, for each E_i, all variables in E_i are substituted by their representant v_i' in the mathematical atoms of μ. All valid atoms (like, e.g., $(v_i - v_i \neq 2), (v_i - v_i > -1)$) are removed, together with all duplicated atoms. If an inconsistent atom is found (like, e.g., $(v_i - v_i = 2), (v_i - v_i \leq -1)$) then MATH-SOLVE terminates returning $Null$.

Second, each remaining atom in the form $(v_i = ...)$ in μ is removed, by applying equality propagation. Throughout this phase, all valid and duplicated atoms are removed, and, if an inconsistent atom is found, then MATH-SOLVE terminates returning $Null$.

L_2: Minimal path plus negative cycle detection. If only atoms in the form $(v_i - v_j \bowtie c)$ are left, $\bowtie \in \{>, <, \geq, \leq\}$, then the resulting problem is solved by invoking a minimum path algorithm with cycle detection, a variant of the Bellman-Ford algorithm described in [8], which either returns a satisfying interpretation for the variables v_i's or verifies there is none. In the former case

MATH-SOLVE decodes back the resulting interpretation and returns it, otherwise it returns $Null$. The algorithm is worst-case quadratic in time and linear in size.

L_3: Linear programming. Otherwise —unless some negated equality $(t_i \neq t_j)$ exist — a linear programming (LP) simplex algorithm is invoked, which, again, either returns a satisfying interpretation for the variables v_i's or verifies there is none. MATH-SOLVE behaves as in the previous case. This algorithm is worst-case exponential in time (but it is well-known that it exhibits polynomial behavior in non-pathological practical cases) and always requires polynomial memory.

L_4: Handling negated equalities. Neither minimal path nor LP procedures handle negated equality constraints like $(t_i \neq t_j)$. In many significant cases — including the ones of practical interest for us, see Section 4— it is always possible to avoid them.

However, in order to preserve expressiveness our design handles them. A trivial way to handle the problem is to split every negated equalities $(t_i \neq t_j)$ into the disjunction of the corresponding strict inequalities $(t_i > t_j) \vee (t_i < t_j)$, and handle the distinct problems separately. This is, of course, rather inefficient.

Instead, we first ignore negated equalities and run one of the algorithms on the remaining part. (i) if there is no solution, then the problem is unsatisfiable anyway, so that it is returned $Null$; (ii) if a solution $\mathcal{I}$ is found, then it is checked against the negated equalities: if it does not contradict them, then $\mathcal{I}$ is returned; (iii) if not, the negated equalities are split and the resulting problems are analyzed.

Notice that the latter event is extremely rare, for two reasons. First, MATH-SOLVE finds a solution at most once in the whole computation. All the other problems are unsatisfiable. Second, a constraint like $(t_i \neq t_j)$ covers only a null-measuring portion of the space of the variables in t_1 and t_2, while the space of the solutions of a solvable linear problem is typically a polyhedron containing infinitely many solutions. Thus $(t_i \neq t_j)$ makes a solvable problem unsolvable only if the solution space degenerates into a n-dimensional point.

3.4 Improvements & Optimizations

We describe some improvements and optimizations for MATH-SAT, some of which come from adapting to our domain improvements and optimizations of the DPLL-based procedures for modal logics [12,14,13] and for temporal reasoning and resource planning [2,19].

Preprocessing atoms. One potential source of inefficiency for the procedure of Figure 1 is the fact that semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other] and thus they may be assigned different [resp. identical] truth values. This causes the undesired generation of a potentially very big amount of intrinsically unsatisfiable assignments (for instance, up to $2^{Atoms(\varphi)-2}$ assignments of the kind $\{(v_1 < v_2), (v_1 \geq v_2), ...\}$).

To avoid these problems, it is wise to preprocess atoms so that to map semantically equivalent atoms into syntactically identical ones:

- *exploit associativity* (e.g., $(v_1 + (v_2 + v_3))$ and $((v_1 + v_2) + v_3)) \Longrightarrow (v_1 + v_2 + v_3)$);
- *sorting* (e.g., $(v_1 + v_2 \leq v_3 + 1)$, $(v_2 + v_1 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1)$);
- *exploiting negation* (e.g., $(v_1 < v_2)$, $(v_1 \geq v_2) \Longrightarrow (v_1 < v_2)$, $\neg(v_1 < v_2)$).

Early Pruning (EP). If an assignment μ' is unsatisfiable, then all its supersets are unsatisfiable. If the unsatisfiability of an assignment μ' is detected during its recursive construction, then this prevents checking the satisfiability of all the up to $2^{|Atoms(\varphi)|-|\mu'|}$ truth assignments which extend μ'.

This suggests to introduce an intermediate satisfiability test on incomplete assignments just before the "split" step:

```
if Likely-Unsatisfiable(μ)              /* early pruning */
    if (Math-Solve(μ) = Null)
        then return False;
```

If the heuristic *Likely-Unsatisfiable* returns $True$, then Math-Solve is invoked on the current assignment μ. If Math-Solve(μ) returns $Null$, then all possible extensions of μ are unsatisfiable, and therefore Math-DPLL returns $False$ and backtracks, avoiding a possibly big amount of useless search.

In this case Math-Solve needs not returning explicitly the interpretation $\mathcal{I}$, so that it can avoid decoding back the solution found by the solver. Moreover, negated equalities $(t_i \neq t_j)$, if any, can be ignored here, as they may only make a satisfiable problem unsatisfiable, not vice versa.

Example 2. Consider the formula φ of Example 1. Suppose that, in four recursive calls, Math-DPLL builds, in order, the intermediate assignment:

$$\mu' = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg(3v_1 - v_3 \leq 6)\}. \qquad (4)$$

(rows 1, 2, 3 and 4 of φ), which contains the conflict set (2) and is thus unsatisfiable. If Math-Solve is invoked on μ', it returns $Null$, and Math-DPLL backtracks without exploring any extension of μ'. ◇

Likely-Unsatisfiable avoids invoking Math-Solve when it is very unlikely that, since last call, the new literals added to μ' can cause inconsistency. (For instance, when they are added only literals which either are purely-propositional or contain new variables.)

Enhanced early pruning (EEP). In early pruning, the call to Math-Solve is not effective if μ is satisfiable. Anyway such a call can produce information which can be used to reduce search afterwords. In fact, the mathematical analysis of μ performed by Math-Solve can allow to assign deterministically truth values to some mathematical atoms $\psi \notin \mu$, and this information can be returned by Math-Solve as a new assignment η, which is unit-propagated away by Math-DPLL.

For instance, assume that all the following mathematical atoms occur in the math-formula. If $(v_1 - v_2 \leq 4) \in \mu$ and $(v_1 - v_2 \leq 6) \notin \mu$, then Math-Solve can

derive deterministically that the latter is true, and thus return an assignment η containing $(v_1 - v_2 \leq 6)$. Similarly, if $(v_1 - v_2 > 2) \notin \mu$ and MATH-SOLVE(μ) finds that v_1 and v_2 belong to the same equivalence class, then it η contains $\neg(v_1 - v_2 > 2)$.

(Mathematical) Backjumping (BJ). An alternative optimization starts from the same observations as those of early pruning. Any branch containing a conflict set is unsatisfiable. Thus, suppose MATH-SOLVE is modified to return also a conflict set η causing the unsatisfiability of the input assignment μ. (As for L_2, a negative cycle represents a conflict set; as for L_3, a technique for returning a conflict sets in LP is hinted in [19].) If so, MATH-DPLL can jump back in its search to the deepest branching point in which a literal $l \in \eta$ is assigned a truth value, pruning the search space below.

Notice the difference w.r.t. early pruning. Both prune the search tree under a branch containing a conflict set. On one hand, backjumping invokes MATH-SOLVE only at the end of the branch, avoiding useless calls. On the other hand, early pruning prunes the search as soon as there is one conflict set in the assignment, whilst backjumping can prune a smaller search tree, as the conflict set returned by MATH-SAT is not necessarily the one which causes the highest backtracking.

Example 3. Consider the formula φ and the assignment μ of Example 1. Suppose that MATH-DPLL generates μ following the order of occurrence within φ, and that MATH-SOLVE(μ) returns the conflict set (2). Thus MATH-DPLL can backjump directly to the branching point $\neg(3v_1 - v_3 \leq 6)$ without branching first on $(v_3 = 3v_5 + 4)$ and $\neg(2v_2 - v_3 > 2)$, obtaining the same pruning effect as in example 2. If instead MATH-SOLVE(μ) returns the conflict set (3), forcing a branch on $(v_3 = 3v_5 + 4)$. $\diamond$

(Mathematical) Learning. When MATH-SOLVE returns a conflict set η, the clause $\neg\eta$ can be added in conjunction to φ. Since then, MATH-DPLL will never again generate any branch containing η.

Example 4. As in Example 3, suppose MATH-SOLVE(μ) returns the conflict set (2). Then the clause $\neg(3v_1 - 2v_2 \leq 3) \vee (2v_2 - v_3 > 2) \vee (3v_1 - v_3 \leq 6)$ is added in conjunction to φ. Thus, whenever a branch contains two elements of (2), then MATH-DPLL will assign the third to $\perp$ by unit propagation. $\diamond$

Learning is a technique which must be used with some care, as it may cause an explosion in size of φ. To avoid this, one has to introduce techniques for discarding learned clauses when necessary [4].

Notice the difference w.r.t. standard boolean backjumping and learning [4]. In the latter case, the conflict set propositionally falsifies the formula, while in our case it is inconsistent from the mathematical viewpoint.

Triggering. This technique is a generalization we propose of a technique adopted in [19]. It comes from the consideration (proved in [16]) that, if we

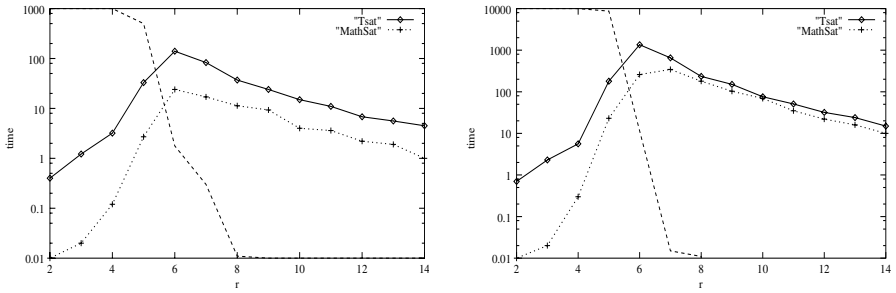


Fig. 2. Comparison between TSAT and MATH-SAT. $k = 2$, $n = 25, 30$, $L = 100$, $r := m/n$ in $[2, ..., 14]$. 100 sample formulas per point. Median CPU times (secs). Background: satisfiability rate.

have mathematical atoms occurring only positively [resp. negatively] in the input formulas, we can drop any negative [positive] occurrence of them from the assignment to be checked by MATH-SOLVE. This is particularly useful when we deal with equality atoms occurring only positively, as it avoids handling negated equalities.

4 Some Experimental Results

We've implemented MATH-SAT in C; MATH-DPLL is built on top of the SIM library [11]; MATH-SOLVE uses alternatively a home-made implementation of Bellman-Ford minimal path algorithm with negative cycle detection [8], and the Simplex LP library LP_SOLVE [5], as described in Section 3.3.

All experiments presented here were run under `Linux RedHat 7.1` on a 4-processor `PentiumIII` 700MHz machine with more than 4GB RAM, with a time limit of 1 hour and a RAM limit of 1GB for each run. (All the math-formulas investigated here are available at `www.science.unitn.it/ rseba/Mathsat.html`, together with our implementation of MATH-SAT.)

4.1 Temporal Reasoning

As a first application example, we consider one of the most studied problems in the domain of temporal reasoning, that of solving the consistency of *disjunctive temporal problems* (DTP). Following [18], we encode the problem as a particular a MATH-SAT problem, where the math-formulas are in the restricted form:

$$\bigwedge_i \bigvee_j (v_{1_{ij}} - v_{2_{ij}} \leq c_{ij}), \tag{5}$$

$v_{k_{ij}}$ and c_{ij} being real variables and integer constants respectively. Notice that here (i) there are no boolean variables (ii) constraints are always in the form $(v_i - v_j \leq c)$ and (iii) they always occur positively.

[18] proposed as a benchmark a random generation model in which DTPs are generated in terms of four integer parameters k, m, n, L: a DTP is produced by randomly generating m distinct clauses of length k of the form (5); each atom is obtained by picking $v_{1_{ij}}$ and $v_{2_{ij}}$ with uniform probability $1/n$ and $c_{ij} \in [-L, L]$ with uniform probability $1/(2L + 1)$. Atoms containing the same variable like $(v_i - v_i \leq c)$ and clauses containing identical disjuncts are discharged.

[2] presented TSAT, a SAT based procedure ad hoc for DTPs like (5) based on Bohm SAT procedure [7] and the Simplex LP library LP_SOLVE [5]. In the empirical testing conducted on the same benchmarks as in [18], TSAT outperformed the procedure of [18]. TSAT is enhanced by a form of forward checking and of static learning, in which it learns binary constraints corresponding to pairs of mutually-inconsistent mathematical atoms.

We have run TSAT and MATH-SAT with enhanced early pruning and mathematical learning on the two hardest problems in [2]. The results are reported in Figure 2. As with TSAT, MATH-SAT curves have a peak around the value of r in which we have a 50% satisfiability rate. MATH-SAT is always faster than TSAT, up to one order of magnitude. Similarly to what happens with the optimizations of TSAT [2], when dropping either enhanced early pruning or mathematical learning in MATH-SAT the performances worsen significantly. Thus, although MATH-SAT is a general-purpose procedure, it turns out to be competitive —and even faster— than a current state-of-the-art specific procedure for this problem.

4.2 Model Checking Properties of Timed Systems

As a second application example, we consider the verification and debugging of properties for timed timed systems (e.g., real-time protocols). In short, a timed system is represented as a timed automaton [1], that is, an automaton augmented by real clock variables x, clock resetting statements in the form $(x := 0)$ and clock constraints in the form $(x \bowtie c)$, $\bowtie \in \{>, <, \geq, \leq\}$. The automaton can perform either instantaneous transitions, which are conditioned by clock constraint and can affect the value of boolean variables and resetting statements, or time elapse transition, which increment all clocks by the same value δ and keeps all values.

In [3] we have extended to timed systems the notion of bounded model checking (BMC) [6] and presented a way to encode such problem into a MATH-SAT problem. Given an automaton A, an LTL property f and an integer bound k, we consider the problem of finding an execution of A of up to length k verifying f, and we encode it into the satisfiability of a CNF math-formula $[[A, f]]_k$, s.t. any interpretation of $[[A, f]]_k$ corresponds to a desired execution path.

We introduce a new real variable z representing the current value of *zero*, and we rewrite every occurrence of a clock x with the difference $x - z$. Then, we replicate the propositional and real variables from 0 to k —e.g., x_i represents the variable x at step i— and "unroll" the transition relation from step 0 to step $k - 1$, so that we have

$$[[A, f]]_k := I_0 \wedge \bigwedge_{i=0}^{k-1} T_{i,i+1} \wedge [[f]]_k. \tag{6}$$

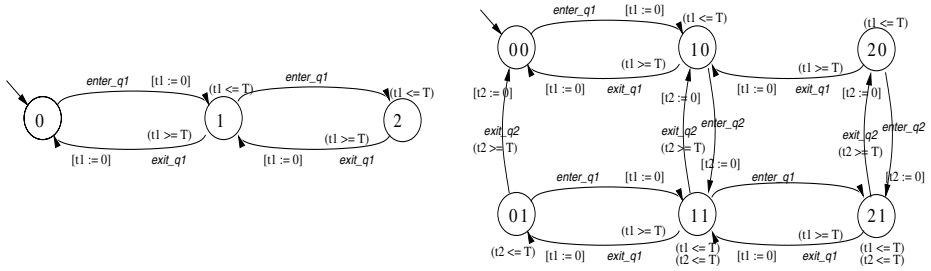


Fig. 3. Timed automata for the post-office problem, for N=1 (left) and N=2 (right).

I_0 is a math formula over the variables at step 0 representing the initial states; $T_{i,i+1}$ is a math formula over the variables at steps i and $i+1$ representing the transition relation; $[[f]]_k$ is a math formula over all the variables from step 0 to step k representing the condition that the path must verify the LTL formula f. (See [6,3,9] for details.) The mathematical atoms in $[[A, f]]_k$ are all of the kind $(x = y)$, $(x_i - z_i = x_j - z_j)$ or $(x - y \bowtie c)$, $\bowtie \in \{\leq, \geq, <, >\}$, such that:

1. every atom in the form $(x_i - z_i = x_j - z_j)$ occurs only positively in $[[A, f]]_k$,
2. for every atom $(x = y)$ in $[[A, f]]_k$, there is a corresponding atom $(x > y)$ in $[[A, f]]_k$ s.t. $\neg(x = y) \rightarrow (x > y)$ is a clause in $[[A, f]]_k$.

We have customized a version of MATH-SAT explicitly for this kind of problems. First, MATH-SOLVE(μ) ignores every negated equality in μ. In fact, by point 1., literals like $\neg(x_i - z_i = x_j - z_j)$ can be ignored because of triggering, and, by point 2., literals like $\neg(x = y)$ can be ignored because they are subsumed by $(x > y)$. Second, following [10,17], the encoder provides some semantic information on the boolean variable, so that the heuristic function *choose_literal()* of Figure 1 can split first on variables labeling transitions, and in order of their step index.

In [3] we presented some empirical tests on timed protocol verification, in which the approach of encoding such problems into math-formulas and running MATH-SAT over them turned out to be competitive wrt. using state-of-the-art verification systems. There the focus was on the encoding, and the goal was to show the effectiveness of our approach w.r.t. other verification approaches. Here instead we focus specifically on the effectiveness of MATH-SAT in solving math-formulas. As we are not aware of any existing decision procedure able to handle efficiently math-formulas of this kind, we restrict to showing how the various variants and optimizations affect the efficiency of MATH-SAT on this kind of formulas on an example.

Consider a post office with N desks, each desk serving a customer every T seconds. Every new customer chooses the desk with shorter queue and, when more than one queue has minimal length, the minimal queue with the minimum index. It is not possible to change a queue after entering it. We want to prove that, although all desks have the same serving time T and customers are "smart", one customer can get into the annoying situation of finding himself in queue

Table 1. MATH-SAT CPU times for the Post-office problem with various optimizations

$k\backslash N$	2	3	4	5
2	0.01	0.05	0.24	1.29
3	0.02	0.08	0.37	2.01
4	0.03	0.10	0.51	2.92
5	0.03	0.13	0.62	5.07
6	0.04	0.17	0.90	8.96
7	0.04	0.24	1.82	37
8	0.05	0.41	4.59	231
9		0.92	16	950
10		0.99	72	≥1h
11			302	≥1h
12			592	≥1h
13				≥1h
14				≥1h
Σ	0.22	3.09	991	≥5h

Basic

2	3	4	5
0.01	0.05	0.25	1.29
0.01	0.08	0.35	2.01
0.02	0.09	0.50	2.95
0.03	0.13	0.68	5.06
0.03	0.16	0.90	8.80
0.05	0.24	1.80	35
0.05	0.41	4.54	230
	0.90	15	945
	0.98	73	≥1h
		299	≥1h
		592	≥1h
			≥1h
			≥1h
0.20	3.04	990	≥5h

Basic+BJ

2	3	4	5
0.01	0.04	0.23	1.31
0.02	0.07	0.36	1.91
0.02	0.10	0.51	2.92
0.03	0.13	0.65	5.00
0.03	0.17	0.88	8.80
0.05	0.23	1.78	37
0.06	0.41	4.53	232
	0.80	14	913
	0.85	68	≥1h
		276	≥1h
		502	≥1h
			≥1h
			≥1h
0.22	2.80	870	≥5h

Basic+EP

2	3	4	5
0.10	0.05	0.25	1.28
0.01	0.08	0.38	2.01
0.02	0.10	0.51	2.83
0.03	0.14	0.68	5.08
0.04	0.18	0.89	8.90
0.05	0.25	1.83	36
0.06	0.40	4.64	229
	0.86	15	916
	0.83	70	≥1h
		288	≥1h
		529	≥1h
			≥1h
			≥1h
0.22	2.89	911	≥5h

Basic+EEP

Table 2. MATH-SAT CPU times for the Post-office problem with customized *choose_literal* and various optimizations

$k\backslash N$	2	3	4	5
2	0.01	0.05	0.25	1.28
3	0.01	0.08	0.38	1.98
4	0.02	0.10	0.49	2.65
5	0.02	0.13	0.63	3.48
6	0.03	0.16	0.74	4.27
7	0.04	0.19	0.88	5.21
8	0.04	0.24	1.09	6.46
9		0.36	1.48	8.98
10		0.29	2.78	16
11			8.43	42
12			5.10	159
13				685
14				208
Σ	0.17	1.60	22.2	1145

Basic

2	3	4	5
0.01	0.05	0.24	1.29
0.02	0.07	0.37	1.98
0.02	0.10	0.49	2.65
0.03	0.13	0.64	3.45
0.03	0.16	0.75	4.22
0.05	0.19	0.91	5.13
0.03	0.23	1.10	6.43
	0.36	1.44	9.08
	0.29	2.81	16
		8.45	42
		5.03	155
			742
			207
0.2	1.58	22.2	1199

Basic+BJ

2	3	4	5
0.01	0.05	0.24	1.30
0.01	0.07	0.36	2.05
0.02	0.10	0.50	2.73
0.03	0.13	0.62	3.51
0.03	0.16	0.76	4.33
0.04	0.19	0.87	5.19
0.05	0.22	1.07	6.40
	0.28	1.28	8.15
	0.28	1.83	11
		2.89	19
		2.07	47
			115
			38
0.19	1.48	12.5	265

Basic+EP

2	3	4	5
0.01	0.05	0.24	1.30
0.01	0.07	0.37	1.97
0.02	0.10	0.49	2.68
0.03	0.13	0.61	3.48
0.03	0.15	0.78	4.26
0.04	0.19	0.90	5.02
0.05	0.23	1.07	6.10
	0.28	1.29	7.54
	0.28	1.80	9.97
		2.94	15
		1.80	33
			80
			23
0.19	1.48	12.3	194

Basic+EEP

after one person, whilst all other queues are empty, and having to wait for a non-instantaneous period in this situation.

The corresponding timing automata for $N = 1$ and $N = 2$ are represented in Figure 3. Each location is labeled by N integers $l_1 l_2 ... l_N$, representing respectively the lengths of queues 1, 2, ... , N. For each queue i a clock variable t_i counts the serving time of the currently served customer. The property is encoded as

$$(20...0)_{k-1} \wedge (20...0)_k \wedge (\delta_{k-1} > 0)$$

for some step $k > 1$, that is, the system is in location 20...0 at both steps $k-1$ and k and a non-null amount of time δ_{k-1} elapses between these two steps. We fix to 2 the maximum queue length of queue 1 and to 1 for the others; this will be enough to show that the problem has a solution, provided the queuing policy of customers. Although the problem is very simple, the size of the automata grow as $3 \cdot 2^N$.

We encoded this problem as described in [3], for increasing values of k and N, and we ran different versions of MATH-SAT on the resulting formulas. The resulting CPU times are reported in Tables 1 and 2. The last row Σ of each table

represents the sum of the values in the corresponding column. Table 2 differs from Table 1 for the fact that in the latter we have used the customized version of *choose_literal()* described above.

All problems are unsatisfiable for $k < 2N + 4$ and satisfiable for $k \geq 2N + 4$, as the minimum path satisfying the property has length $2N + 4$. If we consider the case $N = 2$ of Figure 3, such a path is:

$$00 \overset{enter_{q1}}{\longrightarrow} 10 \overset{enter_{q2}}{\longrightarrow} 11 \overset{enter_{q1}}{\longrightarrow} 21 \overset{\delta=T}{\longrightarrow} 21 \overset{served_{q1}}{\longrightarrow} 11 \overset{enter_{q1}}{\longrightarrow} 21 \overset{served_{q2}}{\longrightarrow} 20 \overset{\delta=T}{\longrightarrow} 20$$

that is, customers C1,C2,C3 enter at time 0; after T seconds three new events occur sequentially within a null amount of time: C1 is served by desk 1, a new customer C4 enters queue 1 (as both queues are of length 1), and C2 is served by desk 2. After this, customers C3 and C4 are in queue 1 while nobody in queuing at desk 2, and C4 will have to wait another T seconds before starting being served.

The CPU times in Tables 1 and 2 suggest the following considerations.

First, MATH-SAT with the customized heuristic *choose_literal()* —which chooses transition variables first in order of their step index— dramatically outperforms the basic version, no matter the other optimizations. In fact, as in [10], initial states and transitions are the only sources of non-determinism: once their values are set, the truth values of all other atoms are either irrelevant or derive deterministically from them. This may drastically restrict the search space. Moreover, as in [17], selecting the transitions in forward [backward] step order allows to avoid selecting transition in intermediate steps whose firing conditions are not reachable from the initial states [resp. from whose consequences the goal states are not reachable].

Second, math backjumping is ineffective on these tests. This is due to the fact that, unlike with DTPs, with these math-formulas the conflict sets returned by MATH-SOLVE are very long and nearly always useless for both math backjumping and learning. In fact, when a (possibly small) conflict set is returned by Bellman-Ford, there is no obvious way to reconstruct back the corresponding minimum conflict set by undoing the variable substitutions over the equivalence classes.

Third, simple and enhanced early pruning improve CPU times slightly in Table 1 and very relevantly in Table 2. In particular, this synergy with the customized heuristics *choose_literal()* is due to the fact that the latter can often choose a "bad" transition whose mathematical prerequisites are not verified in the current state. Without early pruning, this causes the generation of a whole search tree of mathematically inconsistent assignments, whose inconsistency is verified one by one. With early pruning, MATH-DPLL invokes MATH-SOLVE and backtracks just after one sequence of unit propagations.

Fourth, whilst enhanced early pruning seems not faster that simple early pruning in the results of Table 1, a significant improvements appears in Table 2. This synergy with the customized heuristics *choose_literal()* is due to the fact that in many situations the value of a clock x is zero —$(x = z)$ in our encoding— because of either resetting or propagating a previous zero value. If so, performing an enhanced early pruning test before choosing the next transition allows to

falsify all incompatible transition prerequisites on x —like, e.g., $(x - z \geq T)$— and thus to avoid choosing the corresponding "bad" transition.

A final consideration on the effectiveness of our layered architecture arises as soon as we do some profiling: a significant number of MATH-SOLVE calls —about 70% with basic MATH-SAT, 20-30% with EP, about 10% with EEP— are solved directly by propagating equalities (L_1), without calling Bellman-Ford (L_2).

5 Related Work and Conclusions

In this paper we have presented a new approach to the solution of decision problems for combinations of boolean propositions and linear equalities and inequalities over real variables. The approach is general, since it allows to integrate different levels of mathematical solving. This also allows for a significant degree of efficiency, since the more expensive solvers are called only when needed by the subproblem being analyzed. For instance, MATH-SAT is faster than TSAT on the specific class of problems for which the latter has been developed. The other closest related work is [19], where the LPSAT solver is presented, handling math-formulas in which all mathematical atoms occur only positively. The approach, however, is hardwired to the domain of planning, and there is no reference to the architectural issues. In [15], a data structure based on Binary Decision Diagrams (BDDs), combining boolean and mathematical constraints, is used to represent the state space of timed automata. The approach is sometimes very efficient, but it inherits the worst-case exponential memory requirements from BDDs.

In the future, we plan to extend the work presented in this paper along the following directions. First, we will tighten the integration of the different solvers within the SAT architecture. This will allow to incrementally construct the equivalence classes for equality reasoning, and reuse the previously constructed information. Then, we will explore the extensions of the approach to more complex (i.e., quadratic) mathematical constraints, and their applications to formal verification of programs.

References

1. R. Alur. Timed Automata. In *Proc. 11th International Computer Aided Verification Conference*, pages 8–22, 1999.
2. A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proc. European Conference on Planning, ECP-99*, 1999.
3. G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Bounded Model Checking for Timed Systems. Technical Report 0201-05, ITC-IRST, Trento, Italy, January 2002. Submitted for publication.
4. R. J. Bayardo, Jr. and R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT instances. In *Proc AAAI'97*, pages 203–208. AAAI Press, 1997.
5. Michel Berkelaar. The solver lp_solve for Linear Programming and Mixed-Integer Problems. Available at `http://elib.zib.de/pub/Packages/mathprog/linprog/lp-solve/`.

6. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. CAV'99*, 1999.
7. M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.
8. Boris V. Cherkassky and Andrew V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, 1999.
9. A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the Encoding of LTL Model Checking into SAT. In *Proc. 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation*, volume 2294 of *LNCS*. Springer, 2002.
10. E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In *Proc. AAAI'98*, pages 948–953, 1998.
11. E. Giunchiglia, M. Narizzano, A. Tacchella, and M. Vardi. Towards an Efficient Library for SAT: a Manifesto. In *Proc. SAT 2001*, Electronics Notes in Discrete Mathematics. Elsevier Science., 2001.
12. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures – the case study of modal K. In *Proc. CADE13*, LNAI. Springer Verlag, August 1996.
13. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures – the case study of modal K(m). *Information and Computation*, 162(1/2), October/November 2000.
14. I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. of Tableaux'98*, number 1397 in LNAI, pages 27–30. Springer-Verlag, 1998.
15. J. Moeller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. In *Electronic Notes in Theoretical Computer Science*, volume 23. Elsevier Science, 2001.
16. R. Sebastiani. Integrating SAT Solvers with Math Reasoners: Foundations and Basic Algorithms. Technical Report 0111-22, ITC-IRST, November 2001.
17. Ofer Shtrichmann. Tuning SAT Checkers for Bounded Model Checking. In *Proc. CAV'2000*, volume 1855 of *LNCS*. Springer, 2000.
18. K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. In *Proc. AAAI*, pages 248–253, 1998.
19. S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.

Bounded Model Checking for Timed Systems

Gilles AUDEMARD, Alessandro CIMATTI, Artur KORNIŁOWICZ, and Roberto SEBASTIANI.

In *Proceedings of the International Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, pages 243–259. Springer Verlag, 2002.

Bounded Model Checking for Timed Systems*

G. Audemard[1], A. Cimatti[1], A. Kornilowicz[1], and R. Sebastiani[1,2]

[1] ITC-IRST, via Sommarive 16, 38050 Povo, Trento, Italy
{audemard,cimatti,kornilow}@itc.it
[2] DIT, Università di Trento, via Sommarive 14, 38050 Povo, Trento, Italy
rseba@science.unitn.it

Abstract. Enormous progress has been achieved in the last decade in the verification of timed systems, making it possible to analyze significant real-world protocols. An open challenge is the identification of fully symbolic verification techniques, able to deal effectively with the finite state component as well as with the timing aspects. In this paper we propose a new, symbolic verification technique that extends the Bounded Model Checking (BMC) approach for the verification of timed systems. The approach is based on the following ingredients. First, a BMC problem for timed systems is reduced to the satisfiability of a math-formula, i.e., a boolean combination of propositional variables and linear mathematical relations over real variables (used to represent clocks). Then, an appropriate solver, called MATHSAT, is used to check the satisfiability of the math-formula. The solver is based on the integration of SAT techniques with some specialized decision procedures for linear mathematical constraints, and requires polynomial memory. Our methods allow for handling expressive properties in a fully-symbolic way. A preliminary experimental evaluation confirms the potential of the approach.

1 Introduction

The verification of timed systems is a very important and challenging problem. In the last decade, it has being devoted a lot of interest, and significant results have been achieved, making it possible to verify real protocols with limited computational resources (see e.g. [LPY95,DY95]). The verification of timed systems combines the challenge of finite-state variables with the problems related to time: a state can be seen as an assignment to propositional variables and to real variables, called clocks.

The verification of timed systems is traditionally based on the use of Difference Bound Matrices (DBMs) [Dil89], that compactly represent a region associated with an assignment to the clocks that are compatible with a specific assignment to propositional variables. Despite their efficiency, such techniques are basically explicit-state: a complete assignment to propositional variables is associated with a complete region, and the amount of required memory is the most limiting factor in the verification process. Recently proposed techniques, such as DDD [MLAH01], CDDs [LWYP98], and RED [Wan00], provide symbolic representations for the search space. Also in this case,

* This work is sponsored by the CALCULEMUS! IHP-RTN EC project, contract code HPRN-CT-2000-00102, and has thus benefited of the financial contribution of the Commission through the IHP programme.

D.A. Peled and M.Y. Vardi (Eds.): FORTE 2002, LNCS 2529, pp. 243–259, 2002.

however, the memory requirements can be substantial, because of the inheritance of the properties of Binary Decision Diagrams.

In this paper, we propose a new symbolic technique for the verification of timed systems. The approach is a generalization of Bounded Model Checking (BMC) [BCCZ99], that is gaining increasing interest for the verification of finite state systems [Sht00, CFG+01]. The approach consists on encoding the BMC problem of timed systems into the problem of deciding the satisfiability of math-formulas, i.e. boolean combinations of boolean variables and linear (in)equalities over real variables, representing clocks. The resulting problems are then tackled with MATHSAT, a solver combining an efficient procedure for propositional satisfiability (SAT) [DLL62] with mathematical constraint solvers of increasing deductive power.

The approach is rather general, since it allows to express specifications in full LTL, such as fairness properties. Furthermore, the approach is fully symbolic: it allows us to tackle the digital component of timed systems with symbolic technologies as in the untimed case, while the timed component is tackled by means of specialized mathematical constraint solvers. Finally, the math-formulas generated are polynomial w.r.t. the size of the representation of the input system and the maximum path length k, and are solved by a solver requiring a polynomial amount of memory. Although preliminary, our experimental analysis confirms the potentials of the proposed approach.

The paper is organized as follows. In Section 2 we give some background: we recall the basics on timed automata [Alu99], and we describe the MATHSAT problem [ABC+02]. In Section 3 we describe the idea of BMC for timed automata, and we show how to reduce the problem to the satisfiability of a math-formula. In Section 4 we discuss some optimizations to the encoding. In Section 5 we discuss some related approaches. In Section 6 we present some preliminary empirical results. In Section 7 we draw some conclusions, and outline the future directions.

2 Background

2.1 Model Checking Timed Automata

In this section we briefly recall timed automata [Alu99]. An *atomic clock constraint* is any expression in the form $(x \bowtie c)$, $\bowtie \in \{\leq, \geq, <, >\}$, x being a clock variable with values in $\mathbb{R}$ and $c \in \mathbb{Z}$ being a constant; a *clock constraint* is any conjunction of atomic clock constraints. (To this extent, notice that every clock constraint is convex.)

A *timed automaton* A_1 is a tuple $\langle L_1, L_1^0, \Sigma_1, X_1, I_1, E_1 \rangle$ where L_1 is a finite set of locations, $L_1^0 \subseteq L_1$ is the set of initial locations, Σ_1 is a finite set of labels for the possible *events*, X_1 is a finite set of clock variables, I_1 is a map labeling every location $s \in L_1$ with a clock constraint on X_1, and E_1 is the set of switches. Every switch $T = \langle s_i, a, \varphi, \lambda, s_j \rangle$ is characterized by its source and target locations $s_i, s_j \in L_1$, an event $a \in \Sigma_1$, a clock constraint φ on X_1, and a set of clock reset conditions λ in the form $(x = 0)$, $x \in X_1$. If $A_1 = \langle L_1, L_1^0, \Sigma_1, X_1, I_1, E_1 \rangle$ and $A_2 = \langle L_2, L_2^0, \Sigma_2, X_2, I_2, E_2 \rangle$, s.t. X_1 and X_2 are disjoint, then the *product* $A = A_1 || A_2$ is defined as $\langle L, L^0, \Sigma, X, I, E \rangle$, with $L = L_1 \times L_2$, $L^0 = L_1^0 \times L_2^0$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $X = X_1 \cup X_2$, I s.t. $I(s_1, s_2) = I(s_1) \wedge I(s_2)$, and E is defined as follows:

1. if $a \in \Sigma_1 \cap \Sigma_2$, $\langle s_{1i}, a, \varphi_1, \lambda_1, s_{1j}\rangle \in E_1$ and $\langle s_{2i}, a, \varphi_2, \lambda_2, s_{2j}\rangle \in E_2$, then $\langle (s_{1i}, s_{2i}), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s_{1j}, s_{2j})\rangle \in E$;
2. if $a \in \Sigma_1 \backslash \Sigma_2$, $\langle s_{1i}, a, \varphi_1, \lambda_1, s_{1j}\rangle \in E_1$ and $t \in L_2$, then $\langle (s_{1i}, t), a, \varphi_1, \lambda_1, (s_{1j}, t)\rangle \in E$;
3. if $a \in \Sigma_2 \backslash \Sigma_1$, $\langle s_{2i}, a, \varphi_2, \lambda_2, s_{2j}\rangle \in E_2$ and $t \in L_1$, then $\langle (t, s_{2i}), a, \varphi_2, \lambda_2, (t, s_{2j})\rangle \in E$.

The dynamics of the timed automaton A_1 is given by means of a transition system S_{A_1}. A state of S_{A_1} is a pair (s, ν) s.t. $s \in L_1$ and $\nu : X_1 \longmapsto \mathbb{R}^+$ is a *clock evaluation* satisfying $I(s)$. (s, ν) is an initial state iff $s \in L_1^0$ and $\nu(x) = 0$, $\forall x \in X_1$. S_{A_1} can evolve into two different ways. With **time elapse**, if $\delta \geq 0$, then $(s, \nu) \stackrel{\delta}{\longmapsto} (s, \nu + \delta)$, $\nu + \delta$ being the evaluation such that $(\nu + \delta)(x) = \nu(x) + \delta$, $\forall x \in X_1$, and $\nu + \delta'$ satisfies I, $\forall \delta'$ s.t. $0 \leq \delta' \leq \delta$. With **switch**, if $\langle s_i, a, \varphi, \lambda, s_j\rangle \in E_1$ and ν satisfies φ, then $(s_i, \nu) \stackrel{a}{\longmapsto} (s_j, \nu[\lambda = 0])$, $\nu[\lambda = 0]$ being the evaluation assigning $x = 0$ $\forall x \in \lambda$ and agreeing with ν for the other clocks in X_1.

Let $A = \langle L, L^0, \Sigma, X, I, E\rangle$, and let $\mathbf{\Phi}(X)$ and $\mathbf{\Lambda}(X)$ be the set of all possible atomic clock constraints and clock reset conditions on X. Let $\mathbf{\Phi}_A(X)$ be the set of all atomic clock constraints on X occurring in the automata A. We see as atomic propositions the locations $s \in L$ (meaning "A is in location s"), the elements of $\mathbf{\Phi}_A(X)$ and $\mathbf{\Lambda}(X)$. The timed automaton A induces a Kripke structure $\langle S, S^0, R, \mathcal{L}\rangle$, with a finite set of states S, a set of initial states $S^0 \subset S$, a transition relation $R \in S \times S$ and a labeling function $\mathcal{L} : S \longmapsto \mathcal{P}ow(L \cup \mathbf{\Phi}_A(X) \cup \mathbf{\Lambda}(X))$. $\mathcal{L}$ is such that, for every $\sigma \in S$, exactly one $s \in L$ is in $\mathcal{L}(\sigma)$ —we denote s as $location(\sigma)$— and the elements of $\mathbf{\Phi}_A(X) \cup \mathbf{\Lambda}(X)$ in $\mathcal{L}(\sigma)$ have an evaluation in $(\mathbb{R}^+)^{|X|}$. S^0 is the set of $\sigma \in S$ s.t., for every $x \in X$, $(x = 0) \in \mathcal{L}(\sigma)$, $location(\sigma) \in L^0$ and $I(location(\sigma))[X := 0] \subseteq \mathcal{L}(\sigma)$. R is such that, for every $\sigma_i, \sigma_j \in S$, $R(\sigma_i, \sigma_j)$ holds if and only if either (i) one switch $\langle s_i, a, \varphi, \lambda, s_j\rangle$ in Σ is such that $s_i = location(\sigma_i)$, $I(s_i) \subseteq \mathcal{L}(\sigma_i)$, $s_j = location(\sigma_j)$, $I(s_j) \subseteq \mathcal{L}(\sigma_j)$, $\varphi \subseteq \mathcal{L}(\sigma_i)$, $\lambda \subseteq \mathcal{L}(\sigma_j)$ for some $a \in E$; (ii) for some $\delta \geq 0$, $s := location(\sigma_i) = location(\sigma_j)$ is such that $I(s) \subseteq \mathcal{L}(\sigma_i)$, $I(s)[X := X + \delta'] \subseteq \mathcal{L}(\sigma_j)$ for every $\delta' \in [0, \delta]$.

We use Linear Temporal Logic (LTL) with its standard semantics [Eme90] to specify properties of timed automata. Basic propositions are atomic propositions, atomic clock constraint in $\mathbf{\Phi}_A(X)$, and clock reset conditions in $\mathbf{\Lambda}(X)$; a propositional literal (i.e., a basic proposition or its negation) is a LTL formula; if h and g are LTL formulas, then $h \wedge g$, $h \vee g$, $h \leftrightarrow g$, $\mathbf{X}g$, $\mathbf{G}g$, $\mathbf{F}g$, $h\mathbf{U}g$ and $h\mathbf{R}g$ are LTL formulas, $\mathbf{X}$, $\mathbf{G}$, $\mathbf{F}$, $\mathbf{U}$ and $\mathbf{R}$ being the standard "next", "globally", "eventually", "until" and "releases" temporal operators respectively. In order to encompass the case of a property f including atomic clock constraints not in $\mathbf{\Phi}_A(X)$, it is possible to extend the labeling function of the Kripke structure to $\mathcal{L} : S \longmapsto \mathcal{P}ow(L \cup \mathbf{\Phi}_A(X) \cup \mathbf{\Phi}_f(X) \cup \mathbf{\Lambda}(X))$, where $\mathbf{\Phi}_f(X)$ is the set of atomic clock constraints in f.

2.2 Satisfiability of Math-Formulae

We call *math-formula* a boolean combination of boolean variables and linear constraints over numerical variables. We call an *interpretation* a map $\mathcal{I}$ which assigns real and boolean values to real and boolean variables respectively and preserves constant values,

```
boolean MathSAT(formula φ, interpretation & I)
    μ = ∅;
    return MathDPLL(φ, μ, I);

boolean MathDPLL(formula φ, assignment & μ, interpretation & I)
    if (φ == ⊤) {                                          /* base       */
        I = MathSolve(μ) ;
        return (I ≠ Null) ; }
    if (φ == ⊥)                                            /* backtrack */
        return False;
    if {a literal l occurs in φ as a unit clause}          /* unit propagation */
        return MathDPLL(assign(l, φ), μ ∪ {l}, I);
    l = choose-literal(φ);                                 /* split      */
    return    (MathDPLL(assign(l, φ), μ ∪ {l}, I)  or
               MathDPLL(assign(¬l, φ), μ ∪ {¬l}, I) );
```

Fig. 1. Pseudo-code of the basic version of the MATHSAT procedure.

arithmetical and boolean operators. For instance, $\mathcal{I}((x - y \geq 4) \wedge (\neg A_1 \vee (x = y))) = (\mathcal{I}(x) - \mathcal{I}(y) \geq 4) \wedge (\neg \mathcal{I}(A_1) \vee (\mathcal{I}(x) = \mathcal{I}(y)))$. We say that $\mathcal{I}$ *satisfies* a math-formula ϕ, written $\mathcal{I} \models \phi$, iff $\mathcal{I}(\phi)$ evaluates to true. We call *MATHSAT* the problem of checking the satisfiability of a math-formula. We call a *truth assignment* for a math-formula ϕ a truth value assignment μ to (a subset of) the atoms of ϕ. We say that μ *propositionally satisfies* ϕ, written $\mu \models_p \phi$, iff it makes ϕ evaluate to true. We represent truth assignments as sets of literals, with the intended meaning that positive and negative literals represent atoms assigned to true and to false respectively. $\mathcal{I}$ satisfies μ iff it satisfies all its elements. For instance, the assignment $\{(x-y \geq 4), \neg A_1\}$ propositionally satisfies $(x-y \geq 4) \wedge (\neg A_1 \vee (x = y))$, and it is satisfied by $\mathcal{I}$ s.t. $\mathcal{I}(x) = 6, \mathcal{I}(x) = 1$, $\mathcal{I}(A_1) = \bot$.

To solve the MATHSAT problem, we have implemented MATHSAT [ABC$^+$02], a solver based on a variant of the DPLL SAT procedure [DLL62]. The basic schema of such a procedure is reported in Figure 1. MATHSAT takes as input a math-formula φ, expressed in CNF, and (by reference) an empty interpretation $\mathcal{I}$, and returns a truth value asserting whether φ is satisfiable or not, $\mathcal{I}$ being respectively an interpretation satisfying φ or $Null$. MATHSAT invokes MATHDPLL passing as arguments φ and (by reference) an empty assignment μ and the interpretation $\mathcal{I}$. MATHDPLL tries to find a truth assignment μ propositionally satisfying φ which is satisfiable from the mathematical viewpoint. Basically, MATHDPLL is a variant of DPLL, modified to work as an enumerator of truth assignments, whose satisfiability is recursively checked by MATHSOLVE. (The function *assign*(l, φ) assigns l to $\top$ in φ and propositionally simplifies the result.) The key difference w.r.t. standard DPLL is in the "base" step. Standard DPLL needs finding only one satisfying assignment μ, and thus simply returns $True$. MATHDPLL instead also needs checking the satisfiability of μ, and thus it invokes MATHSOLVE(μ). Then it returns $True$ if a non-null interpretation satisfying μ is found, it returns $False$ and backtracks otherwise. MATHSOLVE takes as input an assignment μ and returns either an

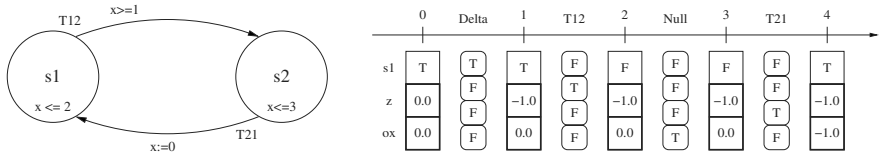


Fig. 2. A simple automaton example

interpretation $\mathcal{I}$ satisfying μ or $Null$ if there is none. In our implementation, MATHSOLVE first performs all the substitutions allowed by the equalities in μ. Then, if only inequalities with two-variable are left, then a variant of Bellman-Ford minimal path algorithm is invoked, a linear programming procedure otherwise. Notice that MATHSAT works in polynomial space. In [Seb01,ABC$^+$02], the proofs of correctness and completeness of MATHSAT are reported, as well as the description of some improvements on the procedure of Figure 1 (e.g. preprocessing and sorting, intermediate assignment checking, triggering, math-driven backjumping and learning) which we have implemented.

3 Bounded Model Checking for Timed Automata

Bounded Model Checking (BMC) is a recent approach to symbolic model checking [BCCZ99]. The starting point is an existential model checking problem $M \models \mathbf{E}f$, for an LTL formula f, and a Kripke structure M. The idea is to solve the problem by looking for a witness to the property that can be presented within a bound of k steps. Given k, the problem is reduced to the satisfiability of a propositional formula $[[M, f]]_k$. If $[[M, f]]_k$ is satisfiable, the propositional model provides a witness of k steps to f. If $[[M, f]]_k$ is unsatisfiable, then nothing can be said about the existence of solutions for $M \models f$ models with higher bound. Thus, the typical technique is to generate and solve $[[M, f]]_k$ for increasing values of k, until either a counter-example is found, or a given time-out is reached. (Completeness can be in principle achieved when k reaches the *diameter* of the problem. Unfortunately, such value is typically hard to compute, and very big.) Despite this limitation, BMC is being increasingly accepted as an effective and practical technique, in particular in the process of falsification, i.e. bug finding. The problem is tackled by refutation, looking for witnesses of bound k to the negation of the property being analyzed. The technique relies on the use of efficient SAT solvers (e.g. based on DPLL procedures) for checking the propositional satisfiability of $[[M, f]]_k$. As shown in [CFG$^+$01], BMC avoids the blow-up in memory that can occur with model checking based on Binary Decision Diagrams, and is therefore able to tackle much larger circuits. Furthermore, SAT-based techniques appear to require less tuning to be effective, and are therefore more amenable to the introduction in industrial settings. In this paper, we address the problem of BMC for $M \models f$ for the case of timed systems, where M is a Kripke structure induced by a timed automaton, and f is an LTL formula. The encoding $[[M, f]]_k$ is a math-formula, where real variables are used to represent the temporal part of the state space and its evolution. The encoding is a combination of a characterization of the paths of the automaton (described in Section 3.1) with a characterization of the paths that satisfy the specification (described in Section 3.2).

$$\bigvee_{s_i \in L_1^0} \underline{s_i} \qquad\qquad \bigwedge_{x \in X_1} (x = z) \tag{1}$$

$$\bigwedge_{s_i \in L_1} (\underline{s_i} \rightarrow \bigwedge_{\psi \in I(s_i)} \psi), \tag{2}$$

Fig. 3. Encoding Initial Conditions and Invariants for A_1

3.1 Encoding Paths and Loops of Timed Automata

In the following, we assume that A_1 and A_2, with $A_i = \langle L_i, L_i^0, \Sigma_i, X_i, I_i, E_i \rangle$, and $A = A_1 || A_2 = \langle L, L^0, \Sigma, X, I, E \rangle$, are given. For explanatory purposes, we use the simple automaton depicted in Figure 2.

Boolean Variables. In order to represent the status and the evolution of the system A_1, we introduce the following propositional variables. For locations, we introduce an array $\underline{s}$ of $\lceil log_2(|L_1|) \rceil$ boolean variables. The intended meaning is that $\underline{s_i}$ holds if and only if the system is in the location s_i. By "$(\underline{s_i} = \underline{s_j})$" we mean "$\bigwedge_n (\underline{s_i}[n] \leftrightarrow \underline{s_j}[n])$", $n \in \{1, ..., \lceil log_2(|L_1|) \rceil\}$. To represent each event $a \in \Sigma_1$, we introduce a boolean variable $\underline{a}$, with the intended meaning that $\underline{a}$ holds if and only if the system executes a switch of event a. For each switch $\langle s_i, a, \varphi, \lambda, s_j \rangle \in E_1$ we introduce a single boolean variable (e.g., T), with the intended meaning that T holds if and only if the system executes the corresponding switch. Finally, we introduce two boolean variables T_δ and T^1_{null}, with the intended meaning that T_δ holds if and only if time elapses by some $\delta > 0$, and that T^1_{null} holds if and only A_1 does nothing, respectively. For instance, in the example of Figure 2, we have the variable $\underline{s}$ for the state, while for transitions we have the variables $T_\delta, T_{12}, T_{21}, T_{null}$. (Since both transitions are labeled by the same event, no variables for events are needed.)

Real Variables. The clocks in the automaton are represented by means of real variables in the encoding, as follows. We introduce a real variable z, called "absolute time reference", whose negated value represents the time elapsed from the beginning of the path being analyzed. Then, for each clock x in the automaton, we introduce an "offset" variable ox whose negated value is the absolute time when the clock was last reset. In general, the value of a clock x is obtained as $ox - z$. In the following, we write r' for the value of the real variable r after a transition of the automaton. In this setting, when the automaton performs a delta transition, and time advances, we have that $z' < z$. Otherwise, time does not elapse, and $z' = z$. Every condition or operation over a clock x can be encoded by means of the difference between the absolute time z and ox. This trivially applies to state constraints and transition constraints of the form $(x \bowtie c)$, with $\bowtie \in \{\leq, \geq, <, >\}$, with $c \in \mathbb{Z}$ being a constant, that are reduced to $(ox' - z' \bowtie c)$. We encode the fact that transition resets a clock by means of the constraint $ox' = z'$. Similarly we impose that clocks have non-negative values by means of the constraint $ox' \leq z'$. In the example of Figure 2, the clock x in the automaton on the left is represented by the difference between ox and z on the right. When clear from context, in the following we write x for ox.

$$\bigwedge_{\substack{T = \langle s_i,\, a,\, \varphi,\, \lambda,\, s_j \rangle \\ T \in E_1}} T \rightarrow \left(\underline{s_i} \wedge \underline{a} \wedge \varphi \wedge \underline{s_j}' \wedge \bigwedge_{x \in \lambda} (x' = z') \wedge \bigwedge_{x \notin \lambda} (x' = x) \wedge (z' = z) \right) \quad (3)$$

$$T_\delta \rightarrow \left((z' - z < 0) \wedge (\underline{s'} = \underline{s}) \wedge \bigwedge_{x \in X_1} (x' = x) \wedge \bigwedge_{a \in \Sigma_1} \neg \underline{a} \right) \quad (4)$$

$$T^1_{null} \rightarrow \left((z' = z) \wedge (\underline{s'} = \underline{s}) \wedge \bigwedge_{x \in X_1} (x' = x) \wedge \bigwedge_{a \in \Sigma_1} \neg \underline{a} \right) \quad (5)$$

$$T^1_{null} \vee T_\delta \vee \bigvee_{T \in E_1} T \quad (6)$$

$$\bigwedge_{\substack{a_k,\, a_r \,\in\, \Sigma_1 \\ a_k \neq a_r}} (\neg \underline{a}_k \vee \neg \underline{a}_r) \qquad \bigwedge_{\substack{T_k = \langle s_i,\, a,\, \varphi_1,\, \lambda_1,\, s_j \rangle, \\ T_r = \langle s_i,\, a,\, \varphi_2,\, \lambda_2,\, s_j \rangle, \\ T_k \neq T_r,\; T_k, T_r \,\in\, E_1}} (\neg T_k \vee \neg T_r) \quad (7)$$

Fig. 4. Encoding Transitions for A_1

Paths and Loops. A path of length k in the (Kripke structure associated with the) automaton is encoded by replicating the propositional and real variables from 0 to k, and the label and transition variables from 0 to $k-1$. (We use the suffix $^{(i)}$ to indicate the step index of a variable: e.g., ${T_\delta}^{(i)}$ and ${T^j_{null}}^{(i)}$ indicate T_δ and T^j_{null} at step i respectively.) A path is obtained by constraining the values of the state vectors at $i, i+1$ via the transition relation of the automaton. Notice that, $z^{(i+1)} \leq z^{(i)}$, $ox^{(i)} \geq z^{(i)}$ and, by induction, $ox^{(i+1)} \leq ox^{(i)}$, as x can either be reset or keep its value. For the example automaton, we depict in Figure 2, on the right, the variables needed to encode a path of bound $k = 4$. The vertical squares represent the state vector at the different steps, where thick squares enclose values for real variables, while the corner-rounded squares represent the propositional values of transitions (from top to bottom $T_\delta, T_{12}, T_{21}, T_{null}$).

A set of implicitly conjoined constraints is needed to make sure that the assignments to the variables represent a legal path of the automaton. The **initial conditions**, holding over the first state vector, state that the system can be only in one of the initial locations (Figure 3, Eq. 1, left), and that the clocks are all zero (right). The **invariants** (Figure 3, Eq. 2), state that if the system is in a location s_i, then all the associated clock constraints must hold, and must be replicated for all state vectors.

The constraints in Figure 4 describe the effect of switches, delta and null transitions, and must be replicated from steps 0 to $k-1$. At step i, the current and state vectors are substituted with the state vector at step i and $i+1$, respectively. Equation 3 encodes **switches** $\langle s_i, a, \varphi, \lambda, s_j \rangle \in E_1$. Intuitively, if $\langle s_i, a, \varphi, \lambda, s_j \rangle$ is being fired, then: (i) before the switch, the automaton is in location s_i, the event a occurs and the constraints φ are verified; (ii) after the switch, the automaton is in location s_j, all clocks in λ are reset, and the values of the other clocks are the same as before the transition; (iii) as no time elapse can occur when switching, the value of z is the same before and after the transition. The formula (4) encodes **delta** transitions: (i) time elapse must be strictly greater than 0, (ii) in the next state the system must be in the same location as in the

current state, (iii) the values of all clocks must be identical to those of the current state, and (iv) no event in Σ_1 can occur together with time elapsing. (The invariants $I(s)$ are conjunctions of linear inequalities, and represent convex regions; thus, if $x' = x$, $z' < z$ and both $x - z$ and $x' - z'$ verify I for every $x \in X_1$, then $x'' - z''$ s.t. $z'' \in [z', z]$ and $x'' = x$ verifies I for every $x \in X_1$.) The formula (5) encodes the **null** transition, enforcing that (i) time elapse must be equal to zero, (ii) in the next state the system must be in the same location as in the current state, (iii) the values of all clocks must be identical to those of the current state, and (iv) no event in Σ_1 can occur.

The remaining formulae (6-7) express the relation between the different transitions. Formula (6) states that at least one variable among the variables T in E_1, T_δ and T^1_{null} must hold, i.e. in system A_1 either a switch shoots, time passes, or stuttering occurs. The formula (7) states mutual exclusion between events, that is, two different events in Σ_1 cannot occur at the same time. The formulas (7) state the mutual exclusion between two switches in the (rare) case they share the same event, source and target locations. In every other case, the mutual exclusion between two switches is a consequence of (3) and the mutual exclusion of states and of events (7).

An infinite, cyclic behaviour can be encoded as a path of length k with a loop back at l, with $0 \leq l < k$. For the propositional part of the state vector, this can be expressed by imposing that each variable has the same value at l and k. For each clock x, we impose that $(ox^{(k)} - z^{(k)} = ox^{(l)} - z^{(l)})$. Notice that it is not possible to express this condition simply by means of equalities between $(ox^{(k)} = ox^{(l)})$, since ox decreases monotonically. Given M and an integer $k \geq 0$, we write $[[M]]_k$ for (the math-formula representing) a path of bound k for the Kripke structure M, ${}_lL_k$ for the loopback condition from k to l.

Product construction. The encoding for the product automaton $A = A_1||A_2$ follows almost directly from the encodings of A_1 and A_2, by conjunction. (Notice that T_δ is common to all systems A_i.) The only addition is the following constraint:

$$\bigwedge_{\substack{a_1 \in \Sigma_1 \setminus \Sigma_2 \\ a_2 \in \Sigma_2 \setminus \Sigma_1}} (\neg \underline{a}_1 \vee \neg \underline{a}_2). \tag{8}$$

needed to prevent two different events $a_1 \in \Sigma_1 \setminus \Sigma_2$ and $a_2 \in \Sigma_2 \setminus \Sigma_1$ to occur at the same time. It is clear that our approach meets the requirement that the combinatorial explosion due to the product construction is not present when generating the encoding. Rather, it is deferred to search time, where it can be tackled by mean of symbolic techniques.

3.2 Encoding LTL Specifications of Timed Automata

The existential BMC problem $M \models_k \mathbf{E}f$, read "there exist an execution path of M of bound k satisfying the LTL property f", is equivalent to the satisfiability problem for the math-formula $[[M, f]]_k$ defined as $[[M]]_k \wedge [[f]]_k$, where $[[f]]_k$ is $[[f]]^0_k \vee \bigvee_{l=0}^{k-1} ({}_lL_k \wedge {}_{l+1}[[f]]^0_k)$. Table 1 describes, in the cases without loopback ($[[f]]^i_k$) and with loopback (${}_l[[f]]^i_k$), the bounded tableau, depending on the structure of the formula. (Without loss of generality we assume that f is in extended negative normal form.) The table encodes the necessary conditions for the existence of a path satisfying the formula. For instance,

Table 1. Recursive definition of $[[f]]^i_k$ and ${}_l[[f]]^i_k$.

f	$[[f]]^i_k$	${}_l[[f]]^i_k$
p	$p^{(i)}$	$p^{(i)}$
$\neg p$	$\neg p^{(i)}$	$\neg p^{(i)}$
$h \wedge g$	$[[h]]^i_k \wedge [[g]]^i_k$	${}_l[[h]]^i_k \wedge {}_l[[g]]^i_k$
$h \vee g$	$[[h]]^i_k \vee [[g]]^i_k$	${}_l[[h]]^i_k \vee {}_l[[g]]^i_k$
$\mathbf{X}g$	$[[g]]^{i+1}_k \; if \; i < k$ $\bot \quad otherwise.$	${}_l[[g]]^{i+1}_k \; if \; i < k$ ${}_l[[g]]^l_k \quad otherwise.$
$\mathbf{G}g$	$\bot$	$\bigwedge^k_{j=min(i,l)} {}_l[[g]]^j_k$
$\mathbf{F}g$	$\bigvee^k_{j=i} [[g]]^j_k$	$\bigvee^k_{j=min(i,l)} {}_l[[g]]^j_k$
$h\mathbf{U}g$	$\bigvee^k_{j=i} \left([[g]]^j_k \wedge \bigwedge^{j-1}_{n=i} [[h]]^n_k \right)$	$\bigvee^k_{j=i} \left({}_l[[g]]^j_k \wedge \bigwedge^{j-1}_{n=i} {}_l[[h]]^n_k \right) \vee$ $\bigvee^{i-1}_{j=l} \left({}_l[[g]]^j_k \wedge \bigwedge^k_{n=i} {}_l[[h]]^n_k \wedge \bigwedge^{j-1}_{n=l} {}_l[[h]]^n_k \right)$
$h\mathbf{R}g$	$\bigvee^k_{j=i} \left([[h]]^j_k \wedge \bigwedge^{j}_{n=i} [[g]]^n_k \right)$	$\bigwedge^k_{j=min(i,l)} {}_l[[g]]^j_k \; \vee$ $\bigvee^k_{j=i} \left({}_l[[h]]^j_k \wedge \bigwedge^{j}_{n=i} {}_l[[g]]^n_k \right) \vee$ $\bigvee^{i-1}_{j=l} \left({}_l[[h]]^j_k \wedge \bigwedge^k_{n=i} {}_l[[g]]^n_k \wedge \bigwedge^{j}_{n=l} {}_l[[g]]^n_k \right)$

in the case of an eventuality formula $\mathbf{F}g$, it requires that g must hold in at least one of the steps within in the bound. Notice that, in the case of a globally formula $\mathbf{G}g$, an infinite behaviour is required in order to be able to provide a witness. The encoding of LTL formulae is very similar to the encoding proposed in [BCCZ99], apart from two differences. First, in the specification, constraints over clocks are also possible as atomic propositions, so that $[[M, f]]_k$ is a math-formula. Second, we define loopback as an equivalence between the state vectors at l and k, while in [BCCZ99] a transition from step k to step l is required.

4 Improvements and Extensions to the Encodings

The encoding described in previous section can be improved and extended in various ways, in order to best exploit the features of the solver. Here we consider some optimizations, and show how they can be implemented in our approach. (Care must be taken since some of them may change the semantics of "next state", and consequently the validity of some LTL properties, in particular those with "$\mathbf{X}$" operators.)

Deterministic propagations of real values. We exploit the fact that the truth value of some mathematical constraint may derive deterministically from those of other mathematical constraints. By making such information explicit in the encoding, the SAT solver deterministically propagates the truth values without investigating the mathematical constraint, and may avoid branching on the truth value of mathematical constraints. This is done by adding the formulas reported in Figure 5 to the encoding. The formulas (9), (10) and (11) make explicit the facts that $z' \leq z$, $x \geq z$ and $x' \leq x$, as observed in Section 3.1, and the fact that equalities and strict inequalities are mutually incompatible. We also propagate the positive and negative values of equalities. If so, for every $x \in X_1$

$$\bigwedge_{x \in X_1} (\neg(x = z) \leftrightarrow (x - z > 0)) \tag{9}$$

$$\neg(z' = z) \leftrightarrow (z' - z < 0) \tag{10}$$

$$\bigwedge_{x \in X_1} \neg(x' = x) \leftrightarrow (x' - x < 0). \tag{11}$$

$$((x = z) \wedge (x' = x) \wedge (z' = z)) \rightarrow (x' = z') \tag{12}$$

$$(\neg(x = z) \wedge (x' = x) \wedge (z' = z)) \rightarrow \neg(x' = z') \tag{13}$$

$$((x = z) \wedge \neg(x' = x) \wedge (z' = z)) \rightarrow \neg(x' = z') \tag{14}$$

$$((x = z) \wedge (x' = x) \wedge \neg(z' = z)) \rightarrow \neg(x' = z') \tag{15}$$

$$((x' = x) \wedge (z' - z < 0) \wedge (x - z > 0)) \rightarrow (x' - z' > 0) \tag{16}$$

$$((z' = z) \wedge \neg(x - z > 0) \wedge (x' - x < 0)) \rightarrow \neg(x' - z' > 0) \tag{17}$$

$$((x - z \bowtie c) \wedge (x' = x) \wedge (z' = z)) \rightarrow (x' - z' \bowtie c) \tag{18}$$

$$(\neg(x - z \bowtie c) \wedge (x' = x) \wedge (z' = z)) \rightarrow \neg(x' - z' \bowtie c) \tag{19}$$

Fig. 5. Formulas for the boolean propagation of mathematical constraints.

we add the formulas (12), (13), (14), (15), (16) and (17). Moreover, we may propagate the positive and negative values of atomic clock constraints when time does not elapse and clocks are not reset, by adding the formulas (18) and (19).

Exploiting parallelism. The formula (8) imposes a mutual exclusion constraint between the two switches T_1 and T_2, since they are labeled by different events. Consider however that T_1 and T_2 do not interfere with each other, since they belong to different automata. Therefore we may want to release such a constraint, by dropping the formula (8) from the encoding, thus allowing them to shoot in parallel. This amounts to allowing the product system A —though not to forcing it— to collapse both sequences $(T_1 \wedge T^2_{null}) \cdot (T^1_{null} \wedge T_2)$ and $(T^1_{null} \wedge T_2) \cdot (T_1 \wedge T^2_{null})$ into one single transition $(T_1 \wedge T_2)$. This results into more compact formulas, and, more importantly, may significantly shorten the length of the minimum counterexample for a given formula, which is essential in the BMC approach. The resulting encoding of $A_1||A_2$ is exactly the conjunction of the encodings of A_1 and A_2.

Forcing System Activity and Compacting Time elapse. Assume that the system is given by the product of the automata $A_0, ..., A_{N-1}$. The encoding allows situations in which all the systems do nothing, that is, all systems A_i execute a transition T^i_{null}. (This corresponds to a time elapse with $\delta = 0$.) To prevent the search engine from considering this situation, we add the formula:

$$\bigvee_{j=0}^{N-1} \neg T^j_{null}. \tag{20}$$

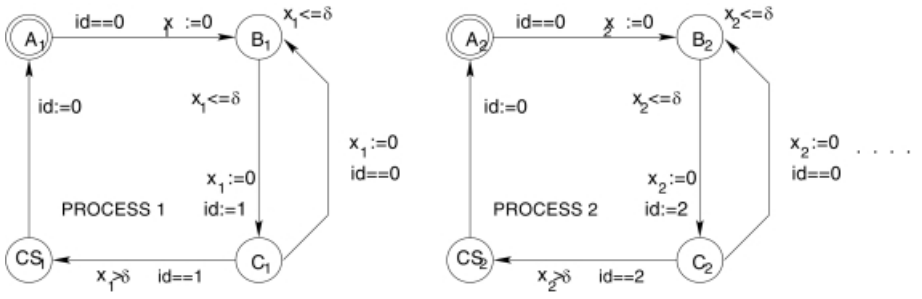


Fig. 6. Two processes in Fischer's mutual exclusion protocol.

Furthermore, we avoid two consequent time elapsing transitions to occur by collapsing $s \overset{\delta}{\longmapsto} s$, $s \overset{\delta'}{\longmapsto} s$ into $s \overset{\delta+\delta'}{\longmapsto} s$. To do this we add the constraint:

$$\neg T_\delta \vee \neg T'_\delta. \tag{21}$$

Adding global variables. Our encoding can be straightforwardly extended to handle global variables v on discrete domains. The intended meaning is that a switch T can be subject to a condition $\psi(\underline{v})$ on the variables v's, and can either assign to v a value n or maintain its value; T_δ maintain the value of v, and T_{null} impose no constraints on v. These facts are encoded respectively as

$$T \rightarrow \psi(\underline{v}), \quad T \rightarrow \underline{v}' = n, \quad T \rightarrow (\underline{v}' = \underline{v}), \quad T_\delta \rightarrow (\underline{v}' = \underline{v}), \tag{22}$$

$\underline{v}$ being a boolean representation of v preserving the mutual exclusion of its values. (Here we assume that (20) is part of the encoding, to make sure that at least one transition states the value of the global variable.)

Exploiting symmetries. Assume that the system is the product of the automata $A_0, ..., A_{N-1}$. We say that the model checking problem is *symmetric* w.r.t. $A_0, ..., A_{N-1}$ if, for every permutation $\sigma = \{k_0 \rightarrow 0, ..., k_{N-1} \rightarrow N-1\}$ of the automata indexes, the permutation of a solution is a solution for the permuted problem. A sufficient condition for symmetry is that the automata $A_0, ..., A_{N-1}$ are identical and that both the initial conditions and the property to be verified are symmetric. If $A_0, ..., A_{N-1}$ are symmetric, we can simplify the search by imposing that, at step i, one of the processes with index $0 \leq j < i$ is forced to fire a transition. We substitute equation 20 with $\bigvee_{j=0}^{min(i,N-1)} \neg T_{null}^{j\ (i)}$, that is, if $i < N-1$, then we drop the disjuncts $\neg T_{null}^{i+1\ (i)} \vee \ldots \vee \neg T_{null}^{N-1\ (i)}$. Notice that, while equation 20 is replicated "as is" for the different time steps, the equation above changes with the step i, so that the i-th instance constraints the possible transitions that can be fired at step i. It is easy to see that the new encoding does not lose any interesting models. It is also clear that a significant amount of search is avoided. The idea is that exponentially-many symmetric executions are collapsed into one. In fact, using $[[M, f]]_k^{symm}$ rather than $[[M, f]]_k$ reduces the space of truth assignments for MATHSAT of up to a $2^{N-1}2^{N-2}...2 = 2^{N(N-1)/2}$ factor.

Given the compositional nature of our encoding, the idea can be generalized to the case in which only *subsets* of $\{A_0, ..., A_{N-1}\}$ are symmetric. For instance, if only $A_0, ..., A_{K-1}$ are symmetric, $K < N$, then we replace (20) with

$$\bigvee_{j=0}^{min(i,K-1)} \neg T_{null}^{j\ (i)} \vee \bigvee_{j=K}^{N-1} \neg T_{null}^{j}. \tag{23}$$

5 Related Work

Our work proposes a new way to tackle the verification of timed systems. In this field, several approaches have been proposed. In [HNSY94], a symbolic approach for formally checking whether a system modeled as a product of timed automata meets its requirements is presented. A product is built from components which can communicate each other through synchronization events. The associated tool KRONOS, is able to perform both backward and forward exploration of the state space, with a symbolic representation combining DBMs ([Dil89]) and BDDs [BDM+98]. UPPAAL ([LPY95]) is a tool for verification of real-time systems. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels and/or shared discrete variables. The description language is a non-deterministic guarded command language with simple data types (e.g. bounded integers, arrays, reals). The model checker is able to check invariants, reachability and some liveness properties by exploring the state space of a system in a symbolic way. To represent the state space, UPPAAL can use Clock Difference Diagrams (CDDs) [LWYP98], or DBMs. Difference Decision Diagrams (DDDs) [MLAH01] are BDD-like data structures to handle boolean formulas over inequalities of the form $x - y < c$ and $x - y \leq c$, and can be used to represent and explore sets of states of a timed system. The associated tool can verify a real time system modeled as a timed guarded command program in a fully symbolic way, by performing reachability analysis and model checking of TCTL formulas using standard fixed-point iteration algorithms. RED [Wan00] is a tool for the verification of real time systems modeled as a set of concurrent processes expressed as timed automata equipped with a clock, discrete variables and pointers. RED is based on the efficient Region Encoding Diagram (RED), a data structure which can be used for fully symbolic model checking of TCTL over timed systems. RED is able to exploit symmetries between processes, and is currently one of the most efficient verification tools in the field. Our approach differs from the above techniques in several respects. On one side, it is limited to the bounded case. On the other side, it allows for the analysis of specifications expressed in LTL, and is therefore able to express general forms of fairness. Furthermore, our approach is based on (an extension of) propositional satisfiability techniques, that are increasingly accepted as an efficient and complementary alternative to the use of Decision Diagrams, for their limited memory requirements. Finally, we use specialized solvers that are able to deal efficiently with different classes of mathematical constraints: equalities (by substitution), binary inequalities (by Bellman-Ford), and arbitrary inequalities (by Symplex). It is also worth mentioning that bounded model checking for timed systems has been

Table 2. Fischer's protocol – reachability (time in seconds, size in MB).

	MATHSAT		MATHSAT,Sym		DDD		Uppal		Kronos		Red		Red,Sym	
N	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	Size
3	0.05	2.9	0.04	2.9	0.11	106	0.01	1.7	0.01	0.8	0.23	2.0	0.19	2.0
4	0.09	3.0	0.08	3.0	0.14	106	0.02	1.9	0.02	2.2	1.00	2.1	0.70	2.1
5	0.20	3.2	0.16	3.2	0.24	106	0.21	1.9	0.09	19	3.70	2.2	2.00	2.4
6	0.60	3.7	0.23	3.7	0.47	106	3.44	6.7	0.39	236	12.00	2.7	5.20	3.1
7	3.20	4.2	0.36	4.2	1.30	106	153	54		-	38	4.0	12	4.7
8	29	4.9	0.52	4.9	3.96	106	-				121	7.6	26	7.8
9	343	5.9	0.75	5.9	14	106					416	16.6	49	13.3
10	3331	6.5	1.01	6.5	62	106					1382	39	90	23
11	-		1.39	7.0	691	106					-		157	38
12			1.89	7.5		-							266	63
13			2.44	8.2									439	100
14			3.24	8.9									709	155
15			4.11	9.7									1118	225
16			5.10	10.7									1717	342
17			6.30	11.7									2582	492
18			8.00	12.9									-	
19			9.50	14.2										

recently (and independently) investigated by other groups [Sor02,PWZ02], the approach closest to ours being [NMA+02].

6 Some Preliminary Empirical Results

In this section, we report some preliminary experimental results, where our approach is used to tackle a case study, and compared with the systems described in previous section. The evaluation is carried out on Fischer's mutual exclusion protocol [Lam87]. N identical processes (described in Figure 6) try to gain access to a critical section CS. The synchronization relies on a global variable id where each process P_i writes its identifier i when entering the waiting state C. Other processes P_j can enter the waiting state C in the same way. If after a certain delay δ still $id = i$, then P_i can enter the critical section CS, from which it eventually exits resetting id to 0; otherwise, as soon as id is reset, it can go back to B, from where it can subsequently retry. This protocol is interesting for many reasons: it is very simple to describe and understand, it contains several elements of interest (e.g. time advance, synchronization, mutual exclusions), it is scalable, so that we can increase its complexity at will by increasing N, and it is symmetric. Despite its simplicity, investigating non-obvious properties is non trivial even with small N's.

To analyze our example, for increasing values of N and increasing values of the bound $k = 1, 2, 3, ...$, we encoded the given problems into math-formulas and we tackled the resulting math-formulas with our implementation of MATHSAT. The experiments were run under Linux RedHat 7.1 on a 4-processor PentiumIII 700MHz machine with more than 6.5GB RAM. The time limit was fixed to 1 hour (only

one processor is allowed for each run), while the memory was limited to 1GB for each run. MATHSAT and all the math-formulas investigated here are available at `http://www.science.unitn.it/ rseba/Mathsat.html`.

As a first example, we have considered the *reachability* problem "Is there a state in which all the processes are in the wait state C", formalized as: $M \models_k \mathbf{EF} \bigwedge_i P_i.C$. For every N, the math-formulas are unsatisfiable for $k \leq N$ and satisfiable for $k > N$. In fact, the shortest solution path has length $N + 1$ (all processes pass from A to B, and then from B to C one at a time). We have compared MATHSAT (without and with the symmetry-exploiting encoding described in Section 4) with the DDD package, with UPPAAL (version 3.2.4), with KRONOS (version 2.4.4), and with RED (version 3.1) (without and with its own symmetry exploitation techniques). The results are collected in Table 2. MATHSATwas run with the default splitting heuristic, i.e. the one of SATZ [LA97]. Times are expressed in seconds and size in megabytes. "-" denotes that the system reached the time or size limit. For MATHSAT, the reported times are *sums* of the times needed to analyze the (unsatisfiable) instances with bound $k = 1, ..., N$ and to the (satisfiable) instance $k = N + 1$.

Although the property is extremely simple, we see that the complexity of the problem blows up quickly with N. Without using the symmetry-exploiting encoding, MATHSAT is better than UPPAAL and KRONOS, slightly worse than RED, worse than DDD and much worse than RED with its symmetry-exploiting technique. With the symmetry-exploiting encoding, MATHSAT runs dramatically better than DDD, UPPAAL and KRONOS, which have no symmetry-exploiting technique, and even better than RED with symmetry-exploiting technique. As memory consumption is concerned, we see that MATHSAT behaves much better than all the other systems.

As a second example, we have considered the following fairness property: "if the i-th process gets infinitely often in B, then it accesses infinitely often in the critical section CS". We look for a counterexample, i.e. we tackle the following problem: $M \models_k \mathbf{E}\neg(\mathbf{GF}P_i.B \rightarrow \mathbf{GF}P_i.CS)$. For every N, the math-formulas are unsatisfiable for $k \leq N + 4$ and satisfiable for $k > N + 4$. In fact, the shortest solution path containing a loop has length $N + 5$: all processes pass from A to B, and then from B to C one at a time; time elapses of a quantity greater than δ; then the last process arrived in C passes alone into CS and then into A; finally they all pass into B again.

To the best of our knowledge, there is no direct way to encode this problem in DDD, UPPAAL, KRONOS and RED. With MATHSAT instead, we can encode it by forcing a loop from step k to each step $l < k$. (We report here only the "interesting" case where $l = 1$). Notice, that the property above is not symmetric, so that this time we cannot use the symmetry-exploiting encoding. The results are collected in Table 3. To emphasize the effects of different splitting heuristics, we run MATHSAT not only with the SATZ heuristics (left), but also with BOEHM's (right). Thus, we notice that SATZ heuristic gives better results with *unsatisfiable* instances, and worse results for satisfiable ones (last entries of each column).

The analysis is clearly preliminary in several respects. First, we are comparing a SAT-based *bounded* model checker with fixpoint-based *unbounded* model checkers, which were not specially conceived for a bounded search. However, as we are not aware of any other bounded model checker for timed system currently available, this

Table 3. Fischer's Protocol – fairness (time in seconds).

	MATHSAT					MATHSAT with Boehm heuristic				
$k \backslash N$	2	3	4	5	6	2	3	4	5	6
2	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.02
3	0.01	0.02	0.01	0.01	0.03	0.01	0.01	0.02	0.03	0.04
4	0.01	0.02	0.02	0.02	0.04	0.01	0.02	0.04	0.07	0.17
5	0.02	0.03	0.05	0.09	0.18	0.01	0.03	0.09	0.3	1.16
6	0.03	0.1	0.21	0.54	1.35	0.02	0.07	0.31	1.52	7.74
7	0.04	0.26	0.97	3.2	9.83	0.02	0.18	1.19	7.14	45
8		0.65	4.8	19.72	70.7		0.06	4.7	33.5	242
9			5.55	112.17	478			0.61	165.9	1348
10				303.17	3086				9.92	7824
11					5002					252
Σ	0.12	1.08	11.62	438.93	8648.15	0.07	0.37	6.98	218.4	9720.13

is not a matter of choice. Second, the analysis can be biased by the fact that we are considering only one case study. Notice however that the case study was not tuned toward SAT-based techniques. (It would indeed be quite easy to find a problem where the data structures for the representation of regions blow up, simply because of the inheritance of the properties of BDDs.) On the contrary, we are tackling an asynchronous system, while so far SAT-based BMC methods have proved particularly effective for synchronous systems. It would be interesting to extend the comparison on synchronous applications such as real-time embedded controllers. It is also important to notice that we are comparing mature approaches, that have been optimized over the years, with a new encoding technique and solver. Having said this, the above results show that our approach as extremely promising. First, as CPU times are concerned, our approach can be comparable with other well-established ones. Second, MATHSAT has much more limited memory requirements. Third, even a very simple exploitation of symmetries can significantly reduce the verification times (and we believe that there are several directions of improvements). Fourth, although bounded, our technique can be used to falsify properties that can not be directly handled by the other approaches.

7 Conclusions and Future Work

In this paper, we have presented a new approach for symbolic model checking of timed systems. We have shown how to encode a BMC problem for timed systems into that of deciding the satisfiability of boolean combinations of boolean variables and atomic linear (in)equalities, which we can solve efficiently by the MATHSAT solver. The approach is fully symbolic, and is not limited to simple reachability, but allows for (Bounded) Model Checking of LTL formulas. Furthermore, the solver can be rather efficient in terms of run-times, even with its limited memory requirements. As BMC in the propositional case, our new technique is intended to be complementary rather than alternative to the current ones, in particular because of its ability of finding (counter)examples, of its expressiveness and of its reduced memory requirements.

In the future, we plan to extend and improve our work along the following directions. First, we want to improve and test new kinds of encodings. In particular, we will investigate alternative representations of locations, events and transitions; we will look for new propagation axioms and invariants to prune search, in particular taking into account more powerful forms of symmetry reduction. Then, we will investigate the customization of MATHSAT for encoded timed automata, by defining splitting heuristics that take into account the different semantics of the variables [GMS98,Sht00] and new mechanisms for propagating and exploiting equalities between real values. Finally, we want to perform an extensive experimental evaluation, also on synchronous domains, to identify the bottlenecks and the strengths of the approach.

References

[ABC+02] G. Audemard, P. Bertoli, A. Cimatti, A. Korni lowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proc. CADE'2002.*, 2002.

[Alu99] R. Alur. Timed Automata. In *Proc. CAV'99*, pages 8–22, 1999.

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, pages 193–207, 1999.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In A. J. Hu and M. Y. Vardi, editors, *Proc. 10th International Conference on Computer Aided Verification, Vancouver, Canada*, volume 1427 of *LNCS*, pages 546–550. Springer-Verlag, 1998.

[CFG+01] F. Copty, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of Bounded Model Checking at an Industrial Setting. In *Proc. CAV'2001*, LNCS. Springer, 2001.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer-Verlag, June 1989.

[DLL62] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with kronos. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 66–75, 1995.

[Eme90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publisher B.V., 1990.

[GMS98] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In *Proc. AAAI'98*, pages 948–953, 1998.

[HNSY94] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[LA97] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, 1997.

[Lam87] L. Lamport. A Fast Mutual-exclusion Algorithm. *ACM Transactions on Computer Systems*, 5(1), 1987.

[LPY95] K. G. Larsen, P. Pettersson, and W. Yi. Model-Checking for Real-Time Systems. In *Fundamentals of Computation Theory*, pages 62–88, 1995.

[LWYP98] K.G. Larsen, C. Weise, W. Yi, and J. Pearson. Clock difference diagrams. Technical Report 98/99, DoCS, Uppsala University, Sweden, 1998.

[MLAH01] J. Moeller, J. Lichtenberg, H. Andersen, and H. Hulgaard. Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams. In *Electronic Notes in Theoretical Computer Science*, volume 23. Elsevier Science, 2001.

[NMA+02] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, and O. Maler. Verification of Timed Automata via Satisfiability Checking. In *Proc. of FTRTFT'02*, LNCS. Springer-Verlag, 2002.

[PWZ02] W. Penczek, B. Woźna, and A. Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Proc. of FTRTFT'02*, LNCS. Springer-Verlag, 2002.

[Seb01] R. Sebastiani. Integrating SAT Solvers with Math Reasoners: Foundations and Basic Algorithms. Technical Report 0111-22, ITC-IRST, November 2001.

[Sht00] Ofer Shtrichmann. Tuning SAT Checkers for Bounded Model Checking. In *Proc. CAV'2000*, volume 1855 of *LNCS*. Springer, 2000.

[Sor02] Maria Sorea. Bounded model checking for timed automata. *ENTCS*, 68(5), 2002.

[Wan00] Farn Wang. Efficient data structure for fully symbolic verification of real-time software systems. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 157–171, 2000.

Bibliographie

[ABC+02a] Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, et Roberto SEBASTIANI. « Integrating Boolean and Mathematical Solving : Foundations, Basic Algorithms, and Requirements ». Dans *Proceedings of the Joint International Conference, AISC 2002 and Calculemus 2002*, pages 231–245, 2002. 51

[ABC+02b] Gilles AUDEMARD, Piergiorgio BERTOLI, Alessandro CIMATTI, Artur KORNIŁOWICZ, et Roberto SEBASTIANI. « A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions ». Dans *Proceedings of the 18th International Conference on Automated Deduction (CADE)*, pages 195–210, 2002. 51

[ABCS05] Gilles AUDEMARD, Marco BOZZANO, Alessandro CIMATTI, et Roberto SEBASTIANI. « Verifying Industrial Hybrid Systems with MathSAT. ». *Electronic Notes in Theorical Computer Science*, 119(2) :17–32, 2005. 55

[ABH+08a] Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, et Lakdhar SAIS. « A Generalized Framework For Conflict Analysis ». Dans *proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 21–27, 2008. . . . . 18, 69

[ABH+08b] Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Said JABBOUR, et Lakhdar SAÏS. « Un cadre général pour l'analyse des conflits ». Dans *Journées Francophones de Programmation par Contraintes(JFPC)*, pages 267–276, 2008. . . . . 18, 69

[ABS00] Gilles AUDEMARD, Belaid BENHAMOU, et Pierre SIEGEL. « AVAL : An enumerative method for SAT ». Dans *Proceedings of first international conference on computational logic (CL)*, pages 373–383, 2000. 8

[ACG99] Alessandro ARMANDO, Claudio CASTELLINI, et Enrico GIUNCHIGLIA. « SAT-Based Procedures for Temporal Reasoning ». Dans *Proceedings of European Conference on Planning*, pages 97–108, 1999. . . . . 49, 51, 54

[ACKS02] Gilles AUDEMARD, Alessandro CIMATTI, Artur KORNIŁOWICZ, et Roberto SEBASTIANI. « Bounded Model Checking for Timed Systems ». Dans *Proceedings of the International Conference on Formal Techniques for Networked and Distributed Systems*, pages 243–259, 2002. 55

[AG93] Alessandro ARMANDO et Enrico GIUNCHIGLIA. « Embedding Complex Decision Procedures Inside an Interactive Theorem Prover ». *Annals of Mathematics and Artificial Intelligence*, 8(3-4) :475–502, 1993. 49

[AGS05] Carlos ANSÓTEGUI, Carla P. GOMES, et Bart SELMAN. « The Achilles' Heel of QBF ». Dans *Proceedings of the Twentieth American National Conference on Artificial Intelligence (AAAI)*, pages 275–281, 2005. 61

[AJS06] Gilles AUDEMARD, Said JABBOUR, et Lakdhar SAIS. « Exploitation des symétries dans les formules booléennes quantifiées ». Dans *journées francophones de la programmation par contraintes (JFPC)*, pages 15–24, 2006. . . . . 41, 69

[AJS07a] Gilles AUDEMARD, Said JABBOUR, et Lakdhar SAIS. « Efficient Symmetry Breaking Predicates for Quantified Boolean Formulae ». Dans *7th International Workshop on Symmetry and Constraint Satisfaction Problems - Affiliated to CP*, 2007. . . . . 41, 44, 69

[AJS07b] Gilles AUDEMARD, Said JABBOUR, et Lakdhar SAIS. « Using SAT-Graph representation to derive hard instances ». Dans *The 14th RCRA workshop Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2007. . . . . 33, 69

[AJS07c] Gilles AUDEMARD, Saïd JABBOUR, et Lakhdar SAIS. « Symmetry Breaking in Quantified Boolean Formulae. ». Dans *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2262–2267, 2007. . . . . 41, 44, 69

[AJS08] Gilles AUDEMARD, Said JABBOUR, et Lakhdar SAIS. « SAT Graph Based Representation : A New Perspective ». *Journal of Algorithms in Cognition, Informatic and Logic*, 63 :17–33, 2008. . . . . 33, 69

[Ake78] Sheldon AKERS. « Binary Decision Diagrams ». *IEEE Transactions on Computers*, 27(6) :509–516, 1978. . . . . 30, 46

[AKS10] Gilles AUDEMARD, George KATSIRELOS, et Laurent SIMON. « A Restriction of Extended Resolution for Clause Learning SAT Solvers ». Dans *24nd Conference on Artificial Intelligence(AAAI)*, pages 15–20, 2010. . . . . 21, 22

[ALM10] Gilles AUDEMARD, Jean-Marie LAGNIEZ, et Bertrand MAZURE. « Approche hybride pour SAT ». Dans *17eme congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle(RFIA)*, pages 279–286, 2010. . . . . 29, 69

[ALMS09a] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Analyse de conflits dans le cadre de la recherche locale ». Dans *Journées Francophones de la Programmation par Contraintes(JFPC)*, pages 215–224, 2009. . . . . 27, 69

[ALMS09b] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Integrating Conflict Driven Clause Learning to Local Search ». Dans *International Workshop on Local Search Techniques in Constraint Satisfaction (affiliated to CP)(LSCS09)*, 2009. 29, 30, 69

[ALMS09c] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Learning in local search ». Dans *21st International Conference on Tools with Artificial Intelligence(ICTAI)*, pages 417–424, 2009. . . . . 27, 69

[ALMS10] Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE, et Lakhdar SAÏS. « Boosting local search thanks to CDCL ». Dans *17th International Conference on Logic for Programming, Artificial Intelligence and Reasoning(LPAR'10)*, pages 474–488, 2010. 29, 69

[Alu99] Rajeev ALUR. « Timed Automata ». Dans *Proceedings of Computer Aided Verification Conference*, pages 8–22, 1999. 55

[AMS04a] Gilles AUDEMARD, Bertrand MAZURE, et Lakdar SAIS. « Dealing with symmetries in Quantified Boolean Formulas ». Dans *proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 257–262, 2004. . . . 41, 43

[AMS04b] Gilles AUDEMARD, Bertrand MAZURE, et Lakdhar SAIS. « Symétries et formules booléennes quantifiées ». Dans *journées nationales des problèmes NP-complets (JNPC)*, 2004. . . . 41, 43

[APT79] Bengt ASPVALL, Michael PLASS, et Robert TARJAN. « A linear-time algorithm for testing the truth of certain quantified Boolean formulas ». *Information Processing Letters*, 8(3) :121–123, 1979. . . . 1, 7, 30, 33

[ARMS02] Fadi ALOUL, Arathi RAMANI, Igor MARKOV, et Karem SAKALLAH. « Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry ». Rapport technique CSE-TR-463-02, University of Michigan, 2002. . . . 41, 44

[ARMS03] Fadi A. ALOUL, Arathi RAMANI, Igor L. MARKOV, et Karem A. SAKALLAH. « Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry ». *Transactions on Computer Aided Design*, 2003. . . . 30, 41, 42, 44

[AS04] Gilles AUDEMARD et Lakdhar SAIS. « SAT Based BDD Solver for Quantified Boolean Formulas ». Dans *proceedings of the 16th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 82–89, 2004. . . . 46

[AS05a] Gilles AUDEMARD et Lakdhar SAIS. « A Symbolic Search Based Approach for Quantified Boolean Formulas ». Dans *proceedings of the eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 16–30, 2005. . . . 46

[AS05b] Gilles AUDEMARD et Lakdhar SAIS. « Une approche symbolique pour les formules booléennes quantifiées ». Dans *journées francophones de la programmation par contraintes (JFPC)*, pages 59–68, 2005. . . . 46

[AS07a] Gilles AUDEMARD et Lakhdar SAIS. « Circuit Based Encoding of CNF Formula ». Dans *proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 16–21, 2007. . . . 31

[AS07b] Gilles AUDEMARD et Laurent SIMON. « GUNSAT : A Greedy Local Search Algorithm for Unsatisfiability. ». Dans *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2256–2261, 2007. . . . 25

[AS08] Gilles AUDEMARD et Laurent SIMON. « Experimenting with Small Changes in Conflict-Driven Clause Learning Algorithms ». Dans *proceedings of Principles and Practice of Constraint Programming, 14th International Conference (CP)*, pages 630–634, 2008. 15

[AS09a] Gilles AUDEMARD et Laurent SIMON. GLUCOSE : a solver that predicts learnt clauses quality. SAT competition booklet, 2009. . . . 17

[AS09b] Gilles AUDEMARD et Laurent SIMON. « Predicting Learnt Clauses Quality in Modern SAT Solvers ». Dans *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 399–404, 2009. . . . 15, 17, 30

[ASMS09] Gilles AUDEMARD, Mouny SAMY-MODELIAR, et Laurent SIMON. « Pourquoi les solveurs SAT modernes se piquent-ils contre des cactus ? ». Dans *Journées Francophones de la Programmation par Contraintes(JFPC)*, pages 245–253, 2009. . . . 15

[Bac02] Fahiem BACCHUS. « Enhancing Davis Putnam with Extended Binary Clause Reasoning ». Dans *Proceedings of the Eighteenth American National Conference on Artificial Intelligence (AAAI)*, pages 613–619, 2002. . . . 26

[BBC+05] Marco BOZZANO, Roberto BRUTTOMESSO, Alessandro CIMATTI, Tommi JUNTTILA, Peter VAN ROSSUM, Stephan SCHULZ, et Roberto SEBASTIANI. « MathSAT : Tight Integration of SAT and Mathematical Decision Procedures ». *Journal of Automated Reasoning*, 35(1-3) :265–293, 2005. 51

[BCCZ99] Armin BIERE, Alessandro CIMATTI, Edmun CLARKE, et Yunshan ZHU. « Symbolic Model Checking without BDDs. ». Dans *Proceedings of International Conference on Tools and Algorithms For The Construction and Analysis of Systems*, pages 193–207, 1999. 8, 55

[BCF+08] Roberto BRUTTOMESSO, Alessandro CIMATTI, Anders FRANZÉN, Alberto GRIGGIO, et Roberto SEBASTIANI. « The MathSAT 4SMT Solver ». Dans *Proceedings of Computer Aided Verification*, pages 299–303, 2008. . . . . 51, 55

[BCM+92] Jerry BURCH, Edmund CLARKE, Kenneth MCMILLAN, David DILL, et L.J. HWANG. « Symbolic Model Checking : 10^{20} States and Beyond ». *Information and Computation*, 98(2) :142–170, 1992. 46

[Ben04] Marco BENEDETTI. « Evaluating QBFs via Symbolic Skolemization ». Dans *Proceedings of Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference*, pages 285–300, 2004. 40

[Ben05a] Marco BENEDDETTI. « sKizzo : A Suite to Evaluate and Certify QBFs ». Dans *cade20*, pages 369–376, 2005. 40

[Ben05b] Marco BENEDETTI. « Quantifier Trees for QBFs ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 378–385, 2005. 38

[BGS99] Laure BRISOUX, Éric GRÉGOIRE, et Lakhdar SAÏS. « Improving Backtrack Search for SAT by Means of Redundancy ». Dans *Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, pages 301–309, 1999. 13

[BGV01] Randal BRYANT, Steven GERMAN, et Miroslav VELEV. « Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic ». *ACM Transactions on Compututational Logic*, 2(1) :93–134, 2001. 51

[BHG09] Adrian BALINT, Michael HENN, et Olivier GABLESKE. « A Novel Approach to Combine a SLS- and a DPLL-Solver for the Satisfiability Problem ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 248–297, 2009. 30

[BHvMW09] Armin BIERE, Marijn J. H. HEULE, Hans van MAAREN, et Toby WALSH, éditeurs. *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. . . . . 1, 197, 203

[Bie04] Armin BIERE. « Resolve and Expand ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 59–70, 2004. 40

[Bie08a] A. BIERE. « PicoSAT essentials ». *Journal on Satisfiability*, 4 :75–97, 2008. 11, 14, 18, 30

[Bie08b] Armin BIERE. « Adaptive Restart Strategies for Conflict Driven SAT Solvers ». Dans *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 28–33, 2008. 14

[Bie09] Armin BIERE. « *Bounded Model Checking* », 14, pages 455–481. Volume 185 of Biere et al. [BHvMW09], 2009. . . . . 1, 5

[BJS97] Roberto J. BAYARDO JR. et Robert C. SCHRAG. « Using CSP Look-Back Techniques to Solve Real-World SAT Instances ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI)*, pages 203–208, 1997. 11

[BKS04] P. BEAME, H. KAUTZ, et A. SABHARWAL. « Towards understanding and harnessing the potential of clause learning ». *Journal of Artificial Intelligence Research*, 22 :319–351, 2004. 21

[BLV07] Marco BENEDETTI, Arnaud LALLOUET, et Jérémie VAUTARD. « QCSP made Practical by virtue of Restricted Quantification ». Dans *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 38–43, 2007. 61

[BM00] Peter BAUMGARTNER et Fabio MASSACCI. « The Taming of the (X)OR ». Dans *Proceedings of first international conference on computational logic CL*, pages 508–522, 2000. 30

[Bou96] Yacine BOUFKHAD. « *Aspects probabilistes et algorithmiques du problème de satisfiabilité* ». Thèse de doctorat, Université de Paris 6, 1996. 31

[BP96] R. BATTITI et M. PROTASI. « Reactive Search, a history-based heuristic for MAX-SAT ». Technical report, Dipartimento di Mathematica, 1996. 24

[Bra01] Ronen BRAFMAN. « A Simplifier for Propositional Formulas with Many Binary Clauses ». Dans *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 515–522, 2001. . . . . 30, 33

[Bry86] Randal E. BRYANT. « Graph-Based Algorithms for Boolean Function Manipulation ». *IEEE Trans. Computers*, 35(8) :677–691, 1986. . . . . 30, 46

[BS94] Belaïd BENHAMOU et Lakhdar SAÏS. « Tractability Through Symmetries in Propositional Calculus ». *Journal of Automated Reasoning*, 12 :89–102, 1994. . . . . 30, 33, 41, 42, 61

[BV02] Randal BRYANT et Miroslav VELEV. « Boolean satisfiability with transitivity constraints ». *ACM Transaction on Compututational Logic*, 3(4) :604–627, 2002. 51

[BW03] Fahiem BACCHUS et Jonathan WINTER. « Effective preprocessing with hyper-resolution and equality reduction ». Dans *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 341–355, 2003. . . . 33, 35

[CG99] Boris CHERKASSKY et Andrew GOLDBERG. « Negative-Cycle Detection Algorithms ». *Mathematical Programming*, 85(2) :277–311, 1999. 53

[CGS98] Marco CADOLI, Andrea GIOVANARDI, et Marco SCHAERF. « An algorithm to Evaluate Quantified Boolean Formulae ». Dans *Proceedings of the Fifteenth American National Conference on Artificial Intelligence (AAAI)*, pages 262–267, 1998. 40

[CHS09] Geoffrey CHU, Aaron HARWOOD, et Peter STUCKEY. « Cache Conscious Data Structures for Boolean Satisfiability Solvers ». *Journal on Satisfiability, Boolean Modeling and Computation*, 6 :99–120, 2009. 11

[CI96] Byungki CHA et Kazuo IWAMA. « Adding new Clauses for Faster Local Search ». Dans *Proceedings of the Thirteenth American National Conference on Artificial Intelligence (AAAI)*, pages 332–337, 1996. . . . . 24, 25, 28

[CJJ+06] David COHEN, Peter JEAVONS, Christopher JEFFERSON, Karen PETRIE, et Barbara SMITH. « Symmetry Definitions for Constraint Satisfaction Problems ». *Constraints*, 11(2-3) :115–137, 2006. 41

[CMFL+06] Sylvie COSTE-MARQUIS, Hélène FARGIER, Jérôme LANG, Daniel Le BERRE, et Pierre MARQUIS. « Representing Policies for Quantified Boolean Formulae ». Dans *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 286–297, 2006. 38

[Coo71] Stephen A. COOK. « The complexity of theorem-proving procedures ». Dans *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971. 5, 7

[Coo76] Stephen COOK. « A short proof of the pigeon hole principle using extended resolution ». *SIGACT News*, 8(4) :28–32, 1976. . . . . 21, 22

[CR79] Stephen A. COOK et Robert A. RECKHOW. « The Relative Efficiency of Propositional Proof Systems ». *Journal of Symbolic Logoc*, 44(1) :36–50, 1979. 20

[Cra92] James M. CRAWFORD. « A theorical analysis of reasoning by symmetry in first-order logic ». Dans *Proceedings of the AAAI Workshop on Tractable Reasoning*, 1992. 41

[CS88] Vašek CHVÁTAL et Endre SZEMERÉDI. « Many hard examples for resolution ». *Journal of the Association for Computing Machinery*, 35(4) :759–768, 1988. 21

[CS01] Philippe CHATALIC et Laurent SIMON. « Multiresolution for SAT Checking ». *International Journal on Artificial Intelligence Tools*, 10(4) :451–481, 2001. . . . . 8, 27, 40, 46

[Dar04] Adnan DARWICHE. « New Advances in Compiling CNF into Decomposable Negation Normal Form ». Dans *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI)*, pages 328–332, 2004. 30

[DD01] Olivier DUBOIS et Gilles DEQUEN. « A backbone-search heuristic for efficient solving of hard 3–SAT formulae ». Dans *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 248–253, 2001. . . . . 1, 8, 13

[Dec90] Rina DECHTER. « Enhancement Schemes for Constraint Processing : Backjumping, Learning, and Cutset Decomposition ». *Artificial Intelligence*, 41(3) :273–312, 1990. 11

[DG84] Wiliam H. DOWLING et Jean H. GALLIER. « Linear-time Algorithms for Testing Satisfiability of Propositional Horn Formulae ». *Journal of Logic Programming*, 3 :267–284, 1984. 7

[Dil89] David DILL. « Timing assumptions and verification of finite-state concurrent systems. ». Dans *Proceedings of Automatic Verification Methods for Finite State Systems*, volume 407, pages 197–212, 1989. 55

[DLL62] Martin DAVIS, George LOGEMANN, et Donald LOVELAND. « A Machine Program for Theorem Proving ». *Journal of the Association for Computing Machinery*, 5 :394–397, 1962. . . . . 1, 5, 8, 21

[DP60] Martin DAVIS et Hilary PUTNAM. « A Computing Procedure for Quantification Theory ». *Journal of the Association for Computing Machinery*, 7 :201–215, 1960. . . . . 1, 5, 8

[DR94] Rina DECHTER et Irina RISH. « Directional Resolution : the Davis-Putnam procedure revisited ». Dans J. DOYLE, E. SANDEWALL, et P. TORASSI, éditeurs, *Proceedings of*

the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR), pages 134–145, 1994. . . . . 8, 27

[dR02] Leornardo DE MOURA et Harald RUESS. « Lemmas on Demand for Satisfiability Solvers ». Dans *sat02*, pages 244–251, 2002. 51

[dR04] Leonardo DE MOURA et Harald RUESS. « An Experimental Evaluation of Ground Decision Procedures ». Dans *Proceedings of Computer Aided Verification*, pages 162–174, 2004. 51

[EB05] Niklas EÉN et Armin BIERE. « Effective Preprocessing in SAT Through Variable and Clause Elimination ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 61–75, 2005. . . . . 8, 33

[EETW00] Uwe EGLY, Thomas EITER, Hans TOMPITS, et Stefan WOLTRAN. « Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas ». Dans *Proceedings of the Seventeenth American National Conference on Artificial Intelligence (AAAI)*, pages 417–422, 2000. 37

[EGN10] Paolo Marin ENRICO GIUNCHIGLIA et Massimo NARIZZANO. « sQueezeBF : An effective preprocessor for QBFs ». Dans *Proceedings of the Thirteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2010. 41

[ES04] Niklas EÉN et Niklas SÖRENSON. « An Extensible SAT-solver. ». Dans *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 333–336, 2004. . . . . 1, 10, 13, 14, 15, 30

[ESW06] Uwe EGLY, Martina SEIDL, et Stefan WOLTRAN. « A Solver for QBFs in Nonprenex Form ». Dans *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI)*, pages 477–481, 2006. 38

[FG00] Paolo FERRARIS et Enrico GIUNCHIGLIA. « Planning as Satisfiability in Nondeterministic Domains ». Dans *Proceedings of the Seventeenth American National Conference on Artificial Intelligence (AAAI)*, pages 748–753, 2000. 37

[FH07] Lei FANG et Michael S. HSIAO. « A new hybrid solution to boost SAT solver performance ». Dans *Proceedings of the conference on Design, automation and test in Europe (DATE)*, pages 1307–1313. EDA Consortium, 2007. . . . . 25, 28

[FR04] Hai FANG et Wheeler RUML. « Complete Local Search for Propositional Satisfiability ». Dans *Proceedings of the Nineteenth American National Conference on Artificial Intelligence (AAAI)*, pages 161–166, 2004. 25

[Fre95] Jon William FREEMAN. « *Improvements to Propositional Satisfiability Search Algorithms* ». Ph.d. thesis, University of Pennsylvania, 1995. . . . . 8, 13

[GIB09] Alexandra GOULTIAEVA, Vicki IVERSON, et Fahiem BACCHUS. « Beyond CNF : A Circuit-Based QBF Solver ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 412–426, 2009. 38

[GJ79] Michael R. GAREY et David S. JOHNSON. *Computers and Intractability : A Guide to the Theory on NP-completness*. A series of books in the Mathematical Sciences. W. H. Freeman and Company, 1979. . . . . 5, 7

[GKNS] Martin GEBSER, Benjamin KAUFMANN, André NEUMANN, et Torsten SCHAUB. « Conflict-Driven Answer Set Solving ». Dans *ijcai07*, pages 386–392. 30

[Glo89] Fred W. GLOVER. « Tabu search - Part I ». *ORSA Journal of Computing*, 1 :190–206, 1989. 24

[Glo90] Fred W. GLOVER. « Tabu search - Part II ». *ORSA Journal of Computing*, 2 :4–32, 1990. 24

[GMOS05] Éric GRÉGOIRE, Bertrand MAZURE, Richard OSTROWSKI, et Lakhdar SAÏS. « Automatic extraction of functional dependencies ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 122–132, 2005. 8, 30

[GMP06] Éric GRÉGOIRE, Bertrand MAZURE, et Cédric PIETTE. « Extracting MUSes ». Dans *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI)*, pages 387–391, 2006. 28, 60

[GN02] Eugene P. GOLDBERG et Yakov NOVIKOV. « BerkMin : a Fast and Robust SAT-Solver ». Dans *In Proceedings of Design Automation and Test in Europe (DATE)*, pages 142–149, 2002. 13

[GN07] Eugene GOLDBERG et Yakov NOVIKOV. « BerkMin : A fast and robust Sat-solver ». *Journal of Discrete Applied Mathematics*, 155(12) :1549–1561, 2007. 13

[GNT01a] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Backjumping for Quantified Boolean Logic Satisfiability ». Dans *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2001. 40, 44

[GNT01b] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « QuBE : A system for deciding Quantified Boolean Formulas Satisfiability ». Dans *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR)*, pages 364–369, 2001. 40

[GNT04] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Monotone Literals and Learning in QBF Reasoning ». Dans *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 260–273, 2004. 40

[GNT06] 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. 40

[GNT07] Enrico GIUNCHIGLIA, Massimo NARIZZANO, et Armando TACCHELLA. « Quantifier Structure in Search-Based Procedures for QBFs ». *IEEE Transactions on CAD of Integrated Circuits and Systems*, 26(3) :497–507, 2007. 38

[Gol08] Eugene GOLDBERG. « A Decision-Making Procedure for Resolution-Based SAT-Solvers ». Dans *SAT*, pages 119–132, 2008. 25, 28

[GS96] Fausto GIUNCHIGLIA et Roberto SEBASTIANI. « Building Decision Procedures for Modal Logics from Propositional Decision Procedure - The Case Study of Modal K ». Dans *Proceedings of Conference on Automated Deduction*, pages 583–597, 1996. 49, 54

[GS00] Fausto GIUNCHIGLIA et Roberto SEBASTIANI. « Building Decision Procedures for Modal Logics from Propositional Decision Procedures : The Case Study of Modal K(m) ». *Information and Computation*, 162(1-2) :158–178, 2000. 54

[GSK98] Carla P. GOMES, Bart SELMAN, et Henry KAUTZ. « Boosting Combinatorial Search through Randomization ». Dans *Proceedings of the Fifteenth American National Conference on Artificial Intelligence (AAAI)*, pages 431–437. American Association for Artificial Intelligence Press, 1998. 9, 14

[Hak85] Armin HAKEN. « The intractability of resolution ». *Theoretical Computer Science*, 39 :297–308, 1985. . . . . 1, 21

[HBPG08] Philipp HERTEL, Fahiem BACCHUS, Toniann PITASSI, et Allen Van GELDER. « Clause Learning Can Effectively P-Simulate General Propositional Resolution ». Dans *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 283–290, 2008. 21

[HJS09a] Youssef HAMADI, Saïd JABBOUR, et Lakhdar SAIS. « Learning for Dynamic Subsumption ». Dans *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 328–335, 2009. 13

[HJS09b] Youssef HAMADI, Said JABBOUR, et Lakhdar SAIS. « ManySAT : a Parallel SAT Solver ». *Journal on Satisfiability, Boolean Modeling and Computation*, 6 :245–262, 2009. . 14

[HJS09c] Youssef HAMADI, Saïd JABBOUR, et Lakhdar SAÏS. LySAT solver description. SAT competition booklet, 2009. 18

[HK05] Edward A. HIRSCH et Arist KOJEVNIKOW. « UnitWalk : A new SAT solver that uses local serach guided by unit clause elimination ». *Annals of Mathematics and Artificial Intelligence*, 43(1-4) :91–111, 2005. 23

[Hoo94] J. N. HOOKER. « Needed : An empirical science of algorithms ». *Operations Research*, 42 :201–212, 1994. . . . . 15, 59

[Hoo99] Holger H. HOOS. « On the run-time behaviour of stochastic local search algorithms for SAT ». Dans *Proceedings of the Sixteenth American National Conference on Artificial Intelligence (AAAI)*, pages 661–666. American Association for Artificial Intelligence, 1999. 24

[Hoo02] Holger H. HOOS. « An adaptive noise mechanism for walkSAT ». Dans *Proceedings of the Eighteenth American National Conference on Artificial Intelligence (AAAI)*, pages 655–660. American Association for Artificial Intelligence, 2002. . . . . 24, 30

[HPS98] Ian HORROCKS et Peter F. PATEL-SCHNEIDER. « FaCT and DLP ». Dans *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX)*, pages 27–30, 1998. 54

[HS09] HyoJung HAN et Fabio SOMENZI. « On-the-Fly Clause Improvement ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 209–222, 2009. 13

[HTH02] Frank HUTTER, Dave A. D. TOMPKINS, et Holger H. HOOS. « Scaling and Probabilistic Smoothing : Efficient Dynamic Local Search for SAT ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, 2002. 24

[Hua07a] Jimbo HUANG. « A case for simple SAT solvers ». Dans *International Conference on Principles and Practice of Constraint Programming*, pages 839–846, 2007. 9

[Hua07b] Jinbo HUANG. « The Effect of Restarts on the Efficiency of Clause Learning ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2318–2323, 2007. 14

[Jab08] Said JABBOUR. « *De la Satisfiabilité Propositionnelle aux Formules Booléennes Quantifiées* ». PhD thesis, Université d'Artois, 2008. . . . . 17, 18

[JN00] Tommi A. JUNTTILA et Ilkka NIEMELA. « Towards an Efficient Tableau Method for Boolean Circuit Satisfiability Checking ». Dans *in proceedings of first international conference on computational logic (CL)*, pages 553–567, 2000. 30

[JW90] Robert G. JEROSLOW et Jinchang WANG. « Solving Propositional Satisfiability Problems ». *Annals of Mathematics and Artificial Intelligence*, 1 :167–187, 1990. . . . . 8, 13

[KBKF95] Hans KLEINE-BÜNING, Marek KARPINSKI, et Andreas FLÖGEL. « Resolution for quantified boolean formulas ». *Information and computation*, 117(1) :12–18, 1995. 40

[Kri85] B. KRISHNAMURTHY. « Short Proofs for Tricky Formulas ». *Acta Informatica*, 22 :253–275, 1985. 41

[KS92] Henri KAUTZ et Bart SELMAN. « Planning as Satisfiability ». Dans *European Conference on Artificial Intelligence*, pages 359–353, 1992. . . . . 1, 5

[KS03] Henri KAUTZ et Bart SELMAN. « Ten Challenges Redux : Recent Progress in Propositional Reasoning and Search ». Dans *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 1–18, 2003. 29

[KSGS09] Lukas KROC, Ashish SABHARWAL, Carla GOMES, et Bart SELMAN. « Integrating Systematic and Local Search Paradigms : A New Strategy for MaxSAT ». Dans *ijcai09*, pages 544–551, 2009. 62

[KSM10] Hadi KATEBI, Karem SAKALLAH, et Igor MARKOV. « Symmetry and Satisfiability : an update ». Dans *Proceedings of the Thirteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2010. 42

[LA97] Chu Min LI et ANBULAGAN. « Look-Ahead Versus Look-Back for Satisfiability Problems ». Dans *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP)*, pages 341–355, 1997. . . . . 8, 15, 33

[Lam87] Leslie LAMPORT. « A Fast Mutual-exclusion Algorithm ». *ACM Transactions on Computer Systems*, 5(1), 1987. 56

[LB01] Daniel LE BERRE. « Exploiting the real power of Unit Propagation Lookahead ». Dans *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, 2001. . . . . 8, 26

[LBRS09] Daniel LE BERRE, Oliver ROUSSEL, et Laurent SIMON. « The SAT 2009 competition results, does theory meets practice ? », 2009.
`http://www.satcompetition.org/2009/sat09comp-slides.pdf`. 25

[Let02] Reinhold LETZ. « Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas ». Dans *Proceedings of Automated Reasoning with Analytic Tableaux and Related Methods, International Conference*, pages 160–175, 2002. . . . . 40, 44

[Lew78] Harry LEWIS. « Renaming a Set of Clauses as a Horn Set ». *Journal of the Association for Computing Machinery*, 25(1) :134–135, 1978. 7

[Li03] Chu Min LI. « Equivalent literal propagation in the DLL procedure ». *Discrete Applied Mathematics*, 130(2) :251–276, 2003. 30

[LK70] Brian LIN et Shen KERNIGHAN. « An efficient heuristic procedure for partitionning graphs ». *Bell System Technical Journal*, 49(2) :291–307, 1970. 23

[LK73] Brian LIN et Shen KERNIGHAN. « An efficient heuristic algorithm for the traveling-salesman problem ». *Operation Research*, 21 :498–516, 1973. 23

[LM09] Chu Min LI et Felip MANYÀ. « *MaxSAT, Hard and Soft Constraints* », 19, pages 613–631. Volume 185 of Biere et al. [BHvMW09], 2009. 62

[LMS06] Inês LYNCE et João MARQUES-SILVA. « SAT in Bioinformatics : Making the Case with Haplotype Inference ». Dans *Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 136–141, 2006. 5

[LMS08] Florian LETOMBE et João P. MARQUES-SILVA. « Improvements to Hybrid Incremental SAT Algorithms ». Dans *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 161–181, 2008. 30

[LS04] Shuvendu LAHIRI et Sanjit SESHIA. « The UCLID Decision Procedure ». Dans *Proceedings of Computer Aided Verification*, pages 475–478, 2004. 51

[LSZ93] Michael LUBY, Alistair SINCLAIR, et David ZUCKERMAN. « Optimal speedup of Las Vegas algorithms ». *Information Processing Letters*, 47(4) :173–180, 1993. 14

[LT09] Christophe LECOUTRE et Sébastien TABARY. « *Symmetry Breaking* », Constraint Networks, chapitre 12. 2009. 41

[LWZ07] Chu Min LI, Wanxia WEI, et Harry ZHANG. « Combining Adaptive Noise and Look-Ahead in Local Search for SAT ». Dans *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 121–133, 2007. 30

[MAVB10] Hratch MANGASSARIAN, Bao Le ALEX, Ra Goultiaeva Andreas VENERIS, et Fahiem BACCHUS. « Leveraging Dominators for Preprocessing QBF », 2010. 41

[McK90] Brendan D. MCKAY. « nauty user's guide (version 1.5) ». Rapport technique TR-CS90 -02, Computer Science Department, Australian National University, 1990. 42

[Min93] Shin-ichi MINATO. « Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems ». Dans *Proceedings of the 39th annual Design Automation Conference*, pages 272–277, 1993. 46

[MJPL90] Steven MINTON, Marc D. JOHNSTON, Andrew B. PHILIPS, et Philip LAIRD. « Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method ». Dans *Proceedings of the Eighth American National Conference on Artificial Intelligence (AAAI)*, 1990. 23

[MLAH01] J. MOELLER, J. LICHTENBERG, H. ANDERSEN, et H. HULGAARD. « Fully Symbolic Model Checking of Timed Systems using Difference Decision Diagrams ». Dans *Electronic Notes in Theoretical Computer Science*, volume 23, 2001. . . . . 55, 57

[MM00] Fabio MASSACCI et Laura MARRARO. « Logical Cryptanalysis as a SAT Problem Encoding and Analysis of the U.S. Data Encryption Standard ». *Journal of Automated Reasoning*, 24 :165–203, 2000. . . . . 5, 62

[MMZ+01] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG, et Sharad MALIK. « Chaff : engineering an efficient SAT solver ». Dans *Proceedings of the 38th annual Design Automation Conference (DAC)*, pages 530–535. ACM, 2001. 1, 9, 10, 13, 14

[Mor93] Paul MORRIS. « The Break Out Method For Escaping From Local Minima ». Dans *Proceedings of the Eleventh American National Conference on Artificial Intelligence (AAAI)*, pages 40–45, 1993. 24

[MS05] Orly MEIR et Ofer STRICHMAN. « Yet Another Decision Procedure for Equality Logic ». Dans *Proceedings of Computer Aided Verification*, pages 307–320, 2005. 51

[MSG95] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « TWSAT : a new local search algorithm for SAT. Performance and Analysis ». Dans *Proceedings of Constraint Programming Workshop on Solving Really Hard Problems*, pages 127–130, 1995. 24

[MSG97] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Tabu Search for SAT ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI)*, pages 281–285, 1997. 24

[MSG98] Bertrand MAZURE, Lakhdar SAÏS, et Éric GRÉGOIRE. « Boosting Complete Techniques thanks to local search methods ». *Annals of Mathematics and Artificial Intelligence*, 22 :319–331, 1998. . . . . 24, 25, 28

[MSK97] David A. MCALLESTER, Bart SELMAN, et Henry A. KAUTZ. « Evidence for Invariants in Local Search ». Dans *Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI)*, pages 321–326, 1997. 24

[MSS96] João P. MARQUES-SILVA et Karem A. SAKALLAH. « GRASP—a new search algorithm for satisfiability ». Dans *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227. IEEE Computer Society, 1996. . . . . 9, 11

[MSS99] João P. MARQUES-SILVA et Karem A. SAKALLAH. « GRASP : A Search Algorithm for Propositional Satisfiability ». *IEEE Transactions on Computers*, 48(5) :506–521, 1999. 9

[MZ02] Marc MÉZARD et Riccardo ZECCHINA. « The random K-satisfiability problem : from an analytic solution to an efficient algorithm ». *Physical Review*, 66 :56–126, 2002. 23

[Nie06] Robert NIEUWENHUIS. « SAT Modulo Theories : A Possible Connection between SAT and CP ». Dans *International Workshop on the Integration of SAT and CP techniques at CP*, 2006. invited paper. 49, 61

[NO05] Robert NIEUWENHUIS et Albert OLIVERAS. « Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools ». Dans *Proceedings of the International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 23–46, 2005. . . . . 10, 51, 55

[OGMS02] Richard OSTROWSKI, Éric GRÉGOIRE, Bertrand MAZURE, et Lakhdar SAÏS. « Recovering and exploiting structural knowledge from CNF formulas ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 185–199, 2002. . . . . 8, 30

[PD07a] Knot PIPATSRISAWAT et Adnan DARWICHE. « A Lightweight Component Caching Scheme for Satisfiability Solvers ». Dans *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 294–299, 2007. 13

[PD07b] Knot PIPATSRISAWAT et Adnan DARWICHE. « RSat 2.0 : SAT Solver Description ». Rapport technique D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007. . . . . 14, 18

[PD08] Knot PIPATSRISAWAT et Adnan DARWICHE. « A New Clause Learning Scheme for Efficient Unsatisfiability Proofs ». Dans *Proceedings of the Twenty-Third American National Conference on Artificial Intelligence (AAAI)*, pages 1481–1484, 2008. 27

[PD09a] Knot PIPATSRISAWAT et Adnan DARWICHE. « On the Power of Clause-Learning SAT Solvers with Restarts ». Dans *Principles and Practice of Constraint Programming - CP 2009*, 2009. 21

[PD09b] Knot PIPATSRISAWAT et Adnan DARWICHE. « Width-Based Restart Policies for Clause-Learning Satisfiability Solvers ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 341–355, 2009. 14

[PL06] Steven PRESTWICH et Inês LYNCE. « Local Search for Unsatisfiability ». Dans *Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 283–296, 2006. . . . . 25, 26, 27

[POSS06] Lionel PARIS, Richard OSTROWSKI, Pierre SIEGEL, et Lakhdar SAIS. « Computing Horn Strong Backdoor Sets Thanks to Local Search ». Dans *International Conference on tools with Artificial Intelligence*, pages 139–143, 2006. 8

[Pro93] Patrick PROSSER. « Hybrid algorithms for the constraint satisfaction problems ». *Computational Intelligence*, 9(3) :268–299, 1993. 11

[PRSS99] Amir PNUELI, Yoav RODEH, Ofer STRICHMAN, et Michael SIEGEL. « Deciding Equality Formulas by Small Domains Instantiations ». Dans *Proceedings of Computer Aided Verification*, pages 455–469, 1999. . . . . 49, 51

[PT09] Luca PULINA et Armando TACCHELLA. « A Structural Approach to Reasoning with Quantified Boolean Formulas ». Dans *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 596–602, 2009. 41

[PTS07] Duc Nghia PHAM, John THORNTON, et Abdul SATTAR. « Building Structure into Local Search for SAT ». Dans *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2359–2364, 2007. . . . . 23, 30

[Pug93] Jean-Francois PUGET. « On the Satisfiability of Symmetrical Constrained Satisfaction Problems ». Dans *Proceedings of Methodologies for Intelligent Systems, 7th International Symposium, ISMIS*, volume 689 de *Lecture Notes in Computer Science*, pages 350–361, 1993. 41

[Pur84] Paul W. PURDOM. « Solving satisfiability with less searching ». *IEEE transactions on pattern analysis ans machine intelligence*, PAMI-6(4) :510–513, 1984. 31

[PV03] Guoqiang PAN et Moshe VARDI. « Optimizing a BDD-Based Modal Solver ». Dans *Proceedings of Ninetenth International Conference on Automated Deduction (CADE)*, pages 75–89, 2003. . . . . 2, 37

[PV04] Guoqiang PAN et Moshe VARDI. « Symbolic Decision Procedures for QBF ». Dans *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 453–467, 2004. 40

[RDK87] R. REITER et J. DE KLEER. « Foundations of assumption-based truth maintenance systems : Preliminary report ». Dans *Proceedings of the Sixth American National Conference on Artificial Intelligence (AAAI)*, pages 183–188, 1987. 46

[RDO85] E. RAMIS, C. DESCHAMPS, et J. ODOUX. *Mathématiques Spéciales*, volume 1, Algèbre. Masson, 1985. 42

[Rin99a] Jussi RINTANEN. « Constructing Conditional Plans by a Theorem-Prover ». *Journal of Artificial Intelligence Research*, 10 :323–352, 1999. . . . . 2, 37

[Rin99b] Jussi RINTANEN. « Improvements to the evaluation of quantified boolean formulae ». Dans *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1192–1197, 1999. 40

[Rob65] John Alan ROBINSON. « A machine-oriented logic based on the resolution principle ». *Journal of the Association for Computing Machinery*, 12 :23–41, 1965. 20

[Rya04] Laurence RYAN. « Efficient algorithms for clause learning SAT solvers ». Master's thesis, Simon Fraser University, 2004. 14

[SB09] Niklas SÖRENSSON et Armin BIERE. « Minimizing Learned Clauses ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 237–243, 2009. 13

[SD05] Carsten SINZ et Edda-Maria DIERINGER. « DPvis - A Tool to Visualize the Structure of SAT Instances ». Dans *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 257–268, 2005. 30

[SDB06] Horst SAMULOWITZ, Jessica DAVIES, et Fahiem BACCHUS. « Preprocessing QBF ». Dans *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 514–529, 2006. 41

[Seb07] Roberto SEBASTIANI. « Lazy Satisability Modulo Theories ». *Journal of Satisfiability*, 3(3-4) :141–224, 2007. 51

[Sho79] Robert SHOSTAK. « A Practical Decision Procedure for Arithmetic with Function Symbols ». *Journal of ACM*, 26(2) :351–360, 1979. 49

[SKC94] Bart SELMAN, Henry A. KAUTZ, et Bram COHEN. « Noise strategies for improving local search ». Dans *Proceedings of the Twelfth American National Conference on Artificial Intelligence (AAAI)*, pages 337–343, 1994. 23

[SKM97] Bart SELMAN, Henry A. KAUTZ, et David A. MCALLESTER. « Ten Challenges in Propositional Reasoning and Search ». Dans *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 50–54, 1997. . 25, 27, 29

[SLM92] Bart SELMAN, Hector J. LEVESQUE, et David MITCHELL. « GSAT : A New Method for Solving Hard Satisfiability Problems ». Dans *Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI)*, pages 440–446, 1992. 23

[SM73] Larry STOCKMEYER et Albert MEYER. « Word Problems Requiring Exponential Time : Preliminary Report ». Dans *Conference Record of Fifth Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973. . . . . 2, 37

[SNC09] Mate SOOS, Karsten NOHL, et Claude CASTELLUCCIA. « Extending SAT Solvers to Cryptographic Problems ». Dans *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 244–257, 2009. . . . . 5, 62

[Soi93] L. H. SOICHER. « GRAPE : a system for computing with graphs and groups ». Dans *"Groups and Computation", DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 11, pages 287–291, 1993. 42

[SP05] Sathiamoorthy SUBBARAYAN et Dhiraj K. PRADHAN. « NiVER : Non Increasing Variable Elimination Resolution for Preprocessing SAT Instances ». Dans *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 276–291, 2005. 33

[Sté06] Igor STÉPHAN. « Boolean Propagation Based on Literals for Quantified Boolean Formulae ». Dans *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI)*, pages 452–456, 2006. 61

[Sté07] Igor STÉPHAN. « Une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées prénexes ». Dans *proceedings des 3emes Journées Francophones de Programmation par Contraintes (JFPC)*, pages 79–88, 2007. . . . . 38, 61

[Sté10] Igor STÉPHAN. « Une nouvelle architecture parallèle pour le problème de validité des QBF ». Dans *proceedings des 6emes Journées Francophones de Programmation par Contraintes (JFPC)*, pages 113–122, 2010. 40

[Str02] Ofer STRICHMAN. « On Solving Presburger and Linear Arithmetic with SAT ». Dans *Proceedings of Formal Methods in Computer-Aided Design, 4th International Conference*, pages 160–170, 2002. 51

[SVV04] Alexander SMITH, Andreas VENERIS, et Anastasios VIGLAS. « Design diagnosis using Boolean satisfiability ». Dans *Proceedings of the Conference on Asia South Pacific Design Automation : Electronic Design and Solution Fair*, pages 218–223, 2004. 1

[SZ05] Haiou SHEN et Hantao ZHANG. « Another Complete Local Search Method for SAT ». Dans *proceedings of Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference*, pages 595–605, 2005. 25

[Sze05] Samir SZEIDER. « Backdoors SET for DLL subsolvers ». *Journal of Automated Reasoning*, 35(1–3) :73–88, 2005. 8

[TBW04] Christian THIFFAULT, Fahiem BACCHUS, et Toby WALSH. « Solving Non-clausal Formulas with DPLL Search ». Dans *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 663–678, 2004. 30

[Tse70] G. TSEITIN. « On the complexity of derivation in propositional calculus ». *Studies in Constructive Mathematics and Mathematical Logic*, 2 :466–483, 1970. 21

[Urq87] Alasdair URQUHART. « Hard examples for resolution ». *Journal of ACM*, 34(1) :209–219, 1987. . . . . 21, 22

[Van96] Robert VANDERBEI. *Linear Programming : Foundations and Extensions*. springer, 1996. 53

[VG05] Allen VAN GELDER. Pool Resolution and Its Relation to Regular Resolution and DPLL with Clause Learning. Dans *proceedings of Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, 40, pages 580–594. 2005. 21

[Wan00] Farn WANG. « Efficient Data Structure for Fully Symbolic Verification of Real-Time Software Systems ». Dans *Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, pages 157–171, 2000. . . . . 55, 57

[WGS03] Ryan WILLIAMS, Carla P. GOMES, et Bart SELMAN. « Backdoors To Typical Case Complexity ». Dans *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1173–1178, 2003. . . . . 1, 7

[WKC10] Sicun Gao WILLIAM KLIEBER, Samir Sapra et Edmund CLARKE. « A Non-Prenex, Non-Clausal QBF Solver with Game-State Learning ». Dans *Proceedings of the Thirteenth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2010. 61

[WW99] Steven WOLFMAN et Daniel WELD. « The LPSAT Engine & Its Application to Resource Planning ». Dans *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 310–317, 1999. . . . . 54, 55

[XHHLB08] Lin XU, Frank HUTTER, Holger H. HOOS, et Kevin LEYTON-BROWN. « SATzilla : portfolio-based algorithm selection for SAT ». *Journal of Artificial Intelligence Research*, 32(1) :565–606, 2008. . . . . 29, 30

[ZM02] Lintao ZHANG et Sharad MALIK. « Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation ». Dans *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP)*, pages 200–215, 2002. 40

[ZM03] Lintao ZHANG et Sharad MALIK. « Cache Performance of SAT Solvers : a Case Study for Efficient Implementation of Algorithms ». Dans *International Conference on Theory and Applications of Satisfiability Testing*, pages 287–298, 2003. 9

[ZMMM01] Lintao ZHANG, Conor F. MADIGAN, Matthew H. MOSKEWICZ, et Sharad MALIK. « Efficient conflict driven learning in a boolean satisfiability solver ». Dans *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 279–285. IEEE Press, 2001. . . . . 9, 11, 12, 16, 17, 30

[ZS96] Hantao ZHANG et Mark E. STICKEL. « An Efficient Algorithm for Unit Propagation ». Dans *Proceedings of Mathematics and Artificial Intelligence Symposium*, 1996. . . . . 8, 9

Résumé

Ce document présente l'ensemble des travaux de recherche réalisé depuis la fin de ma thèse. Ils ont un socle commun, le problème SAT (pour satisfaisabilité). J'ai commencé à travailler sur ce problème, et essentiellement sur l'algorithmique associée, lors de ma thèse. La dichotomie entre la simplicité à le formuler et la difficulté à le résoudre m'a toujours fasciné. J'ai également travaillé sur quelques extensions de ce problème, autorisant une expressivité plus importante comme le problème QBF (Formules Booléennes Quantifiées) ou le problème SMT (SAT Modulo Théories).

Ce document se compose de trois parties. La première d'entre elle est un synthèse des travaux de recherche. Je l'ai séparé en 3 principaux chapitres. Le premier résume mes travaux autour de SAT, le second autour de QBF et enfin le troisième autour de SMT. La seconde partie de ce mémoire est un curriculum vitae présentant mes diverses activités de recherche, d'enseignement, mais également la liste de mes publications. Enfin, la troisième partie est une sélection de mes publications couvrant les divers domaines abordés dans la première partie.

Mots-clés: SAT, CDCL, QBF, SMT