Une restriction de la résolution étendue pour les démonstrateurs SAT modernes*

Gilles Audemard[1] **George Katsirelos**[1] **Laurent Simon**[2]

[1] Univ. Lille-Nord de France – CRIL/CNRS UMR8188 – Lens, F-62307, France
[2] Univ. Paris-Sud – LRI/CNRS UMR 8623 / INRIA Saclay – Orsay, F-91405, France
audemard@cril.fr gkatsi@gmail.com simon@lri.fr

Résumé

La plupart des solveurs SAT modernes se basent, avec succès, sur les mécanismes d'analyse de conflits et d'apprentissage initialement introduits dans les solveurs GRASP et CHAFF. D'un point de vue théorique, ce succès a été partiellement expliqué à l'aide de l'équivalence, en termes de puissance, entre le système de preuves implanté par l'apprentissage (avec redémarrages) et la résolution générale, alors que les précédents démonstrateurs SAT implantent des systèmes de preuve moins puissants. Néanmoins, des bornes inférieures exponentielles subsistent pour la résolution générale, ce qui suggère une voie prometteuse – mais théorique – permettant une amélioration significative des démonstrateurs SAT : l'utilisation de systèmes de preuves plus puissants. Transformer cette voie théorique en améliorations pratiques dans le cas général a cependant toujours échoué.

Dans cet article, nous présentons un solveur SAT qui utilise une restriction de la résolution étendue. Ce solveur améliore les solveurs existant, en pratique, sur des instances issues des dernières compétitions, mais également sur des instances difficiles pour la résolution comme les instances XOR-SAT.

Abstract

Modern complete SAT solvers almost uniformly implement variations of the clause learning framework introduced by Grasp and Chaff. The success of these solvers has been theoretically explained by showing that the clause learning framework is an implementation of a proof system which is as poweful as resolution. However, exponential lower bounds are known for resolution, which suggests that significant advances in SAT solving must come from implementations of more powerful proof systems.

We present a clause learning SAT solver that uses extended resolution. It is based on a restriction of the application of the extension rule. This solver outperforms existing solvers on application instances from recent SAT competitions as well as on instances that are provably hard for resolution, such as XOR-SAT instances.

*Ce travail est supporté par l'ANR « UNLOC », ANR 08-BLAN-0289-01.

1 Introduction

Depuis une vingtaine d'années des progrès spectaculaires ont été réalisé dans la résolution pratique du problème de satisfaisabilité (SAT). Dans le cadre des problèmes structurés (par opposition aux problèmes aléatoires uniformes), l'apprentissage, initialement introduit dans GRASP [11] représente maintenant la brique de base de tout solveur « moderne ». Le passage à l'échelle des instances industrielles repose quant à lui principalement sur l'utilisation des structures de données paresseuses, associés à l'heuristique proposée dans CHAFF [12]. Peu après, en proposant une réécriture simplifiée et compacte, le solveur MINISAT [7] a également aidé à mieux comprendre les divers composants essentiels aux solveurs SAT modernes d'un point de vue pratique. Si on se place maintenant du côté théorique, d'importants progrès ont aussi été obtenus. Ainsi, l'efficacité de ces solveurs a pu être partiellement expliqué d'un point de vue théorique en montrant que l'apprentissage permettait d'atteindre la puissance des systèmes de preuves par résolution générale [2], ce qui n'est pas le cas pour les algorithmes de type DPLL [6, 1]. Ces derniers sont en effet limités à la puissance d'une forme resteinte de la résolution (*tree like resolution*). Il existe ainsi théoriquement des exemples admettant des preuves de réfutation de taille minimale exponentiellement plus longues pour les algorithmes de type DPLL que pour les algorithmes CDCL (*conflict driven, clause learning*), alors

que l'inverse n'est pas vrai.

D'un point de vue pratique, même si les solveurs CDCL sont très efficaces sur des instances issues de la vérification formelle [14], de nouveaux problèmes provenant de domaines comme la cryptographie et la bio-informatique [16] sont régulièrement proposés comme nouveaux challenges à la communauté. Afin de permettre une amélioration pratique significative, nous proposons dans ce travail d'étendre les solveurs CDCL à un système de preuves plus puissant que la résolution. Un candidat naturel pour cela est la résolution étendue (ER) [19]. La résolution étendue ajoute à la résolution une règle simple : la possibilité d'introduire des lemmes à la formule. Malgré son apparente simplicité, ER est aussi efficace que le système de preuve le plus efficace connu actuellement : L'*Extended Frege Systems* [21]. En effet, il n'a toujours pas été identifié de familles d'instances difficiles pour ER (*i.e.* n'admettant que des réfutations « longues »).

Néanmoins, il existe un obstacle majeur à franchir avant d'implanter un solveur basé sur la résolution étendue : comment choisir les bons lemmes à ajouter à la formule ? D'une part les possibilités d'extension sont gigantesques à chaque étape, et d'autre part trouver les bons lemmes est « difficile ». Par exemple, dans [5], Stephen COOK propose une preuve courte du problème des pigeons en utilisant la résolution étendue, mais les lemmes ajoutés tiennent compte de la sémantique du problème. Ainsi, même si ER peut être utilisé pour construire des preuves courtes de problèmes difficiles pour RES, aucun solveur implantant ER n'a encore été proposé. Choisir judicieusement les bons lemmes est au moins aussi difficile que résoudre directement le problème initial. Il est cependant à noter que certains travaux ont déjà, dans une certaine mesure, utilisé avec succès la résolution étendue, mais de manière indirecte seulement. Ainsi, les solveurs basés sur les BDD peuvent être vus comme l'utilisation de ER [18, 4], tout comme l'utilisation des symétries [17]. Le challenge est ici de construire un solveur implantant la résolution étendue et fonctionnant en pratique mieux que les solveurs CDCL, sur leurs problèmes favoris.

Nous proposons dans ce travail une restriction simple à ER qui réduit de manière significative le nombre potentiel de lemmes à ajouter à la formule. Cette restriction est en relation étroite avec le coeur des algorithmes CDCL : l'observation de régularités dans la succession de clauses apprises durant la construction de la preuve d'insatisfaisabilité. Cette approche est bien différente des approches mentionnées préalablement, dans la mesure où la référence à ER est explicite et peut permettre d'obtenir un solveur implantant un système de preuves aussi puissant que ER non restreinte. L'autre caractère novateur de notre approche vient du fait que nous traitons les nouvelles variables ajoutées comme une ressource similaire aux clauses apprises. Nous apprenons beaucoup de nouvelles variables, mais en oublions également beaucoup afin de limiter la consommation mémoire et de maintenir une bonne vitesse de propagation unitaire.

Le reste de l'article est organisé de la manière suivante : nous commençons par rappeler certaines notions nécessaires à la compréhension de l'article : les solveurs CDCL, les systèmes de preuves et la résolution étendue. Nous décrivons ensuite la restriction de la résolution étendue que nous utilisons et montrons comment elle peut améliorer la résolution. Nous continuons en décrivant comment cette restriction peut être efficacement implantée dans un solveur CDCL. Avant de conclure, nous montrons que le solveur obtenu peut améliorer de manière significative les solveurs actuels sur de nombreuses instances difficiles.

2 Rappels

Le problème SAT consiste à trouver une affectation des variables d'une formule propositionnelle ϕ exprimée sous forme normale conjonctive (CNF) afin que toutes les clauses de ϕ soient satisfaites. Nous considérons que la formule ϕ contient n variables propositionnelles $x_1 \dots x_n$. Pour chaque variable x_i, il existe deux *littéraux*, à savoir x_i et $\overline{x}_i$. Nous notons l un litéral arbitraire et $\overline{l}$ sa négation. Une formule CNF est une conjonction de clauses, chaque clause étant une disjonction de littéraux. Par convention, une CNF pourra être vue comme un ensemble de clauses et chaque clause pourra être vue comme un ensemble de littéraux. Nous supposons que le lecteur est familier avec les notions de sauts arrières (*backtracking*) et de de propagation unitaire.

Dans cet article, nous utilisons les solveurs CDCL [11, 12, 7], qui sont des algorithmes complets étendant la fameuse procédure de Davis et Putnam [6]. Ils incorporent des structures de données paresseuses, des sauts arrières, des nogoods des heuristiques dynamiques et des redémarrages. L'agorithme 1 décrit le schéma général d'un solveur CDCL. À chaque itération de la boucle principale la propagation unitaire est effectuée (ligne 3). Si un conflit apparaît (ligne 4), un nogood (également appelé clause assertive) est construit, en utilisant un schéma d'apprentissage basé sur la résolution [23], puis enregistré dans la base des clauses (ligne 5 ; 8). Un saut arrière est effectué jusqu'à ce que le nogood redevienne unitaire et permette un nouveau processus de propagation unitaire (ligne 9). Dans le cas où aucun conflit n'apparait, une nouvelle variable de décision est choisie de manière heuristique (ligne 13–14). La recherche est terminée lorsque toutes les variables sont affectées (lignes 11–12) ou lorsque la clause vide est générée (lignes 6–7).

Algorithm 1 Solveur Conflict Driven - Clause Learning

```
1: procedure CDCL(φ)
2:     loop
3:         Propagation Unitaire
4:         if une clause C est FAUSSE then
5:             C' = AnalyseConflit(C)
6:             if C' = ∅ then
7:                 retourner UNSAT
8:             Enregistrer C'
9:             Retour arrière jusqu'à ce que C' soit unitaire
10:        else
11:            if toutes les variables sont affectées then
12:                retourner SAT
13:            Choisir un littéral l
14:            Affecter l à un nouveau niveau de décision
```

2.1 Systèmes de preuves

Un système de preuves propositionel est un algorithme en temps polynômial SP tel que, pour toute formule propositionnelle insatisfaisable ϕ, il existe une preuve p tel que SP accepte l'entrée (ϕ, p), c'est à dire que SP montre que p est bien une preuve d'insatisfaisabilité pour ϕ. En d'autres termes, un système de preuve est un moyen efficace de vérifier la validité d'une preuve d'insatisfaisabilité (trouver une telle preuve reste cependant 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 croit polynomialement (resp. exponentiellement) avec la taille des instances de $\mathcal{F}$. On dit alors que $\mathcal{F}$ est facile (resp. difficile) pour SP.

Un système de preuves SP_1 p-simule un système de preuves SP_2 si pour toute formule insatisfaisable ϕ la plus petite preuve de ϕ dans SP_1 est au pire polynomialement plus grande que la plus petite preuve de ϕ dans SP_2. S'ils se p-simulent l'un l'autre, ils sont alors polynomialement équivalents. La notion de p-simulation exprime une hiérarchie 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 preuves le plus connu et le plus utilisé est le système de preuves par résolution [15] (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 clauses sont celles de la formule initiale et les clauses suivantes sont produites par résolution. C'est à dire que la clause C_k est générée à partir des clauses C_i et C_j, où $i < j < k$ et il existe un littéral l tel que $C_i \equiv l \vee \alpha$, $C_j \equiv \bar{l} \vee \beta$, $C_k \equiv \alpha \vee \beta$ et $l \in \alpha \implies \bar{l} \notin \beta$. La clause C_k est alors la résolvante de C_i et C_j, ce qui est noté $C_k = C_i \otimes_l C_j$. La clause C_s est la clause vide.

La trace d'exécution d'un solveur CDCL peut se voir comme un système de preuves [2, 22, 9, 13] 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 [6] 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 à ses propres li-

C'est certainement ce qui explique l'absence de la résolution étendue dans les solveurs SAT actuels. À titre d'exemple, dans [5], COOK utilise la sémantique du problème pour choisir les lemmes à ajouter. Dans notre approche, cette difficulté est en partie écartée grâce à une restriction des possibilités d'extension.

3.1 Restriction proposée

Soient deux littéraux l_1 et l_2 provenant de deux variables différentes. Supposons que nous ayons appris durant la recherche les clauses $C_i = \bar{l}_1 \vee \alpha$ et $C_j = \bar{l}_2 \vee \alpha$. Supposons enfin que ces clauses soient toutes deux utilisées dans un processus de résolution avec la même séquence de clauses $C_{m_1}, \ldots, C_{m_k}$ afin de dériver $C'_i = \bar{l}_1 \vee \beta$ et $C'_j = \bar{l}_2 \vee \beta$. Dans ce cas, il peut être judicieux d'étendre la formule initiale avec la nouvelle variable $x \Leftrightarrow l_1 \vee l_2$. Ceci va alors permetttre d'effectuer une résolution entre C_i et C_j avec $\overline{x} \vee l_1 \vee l_2$ afin d'obtenir $C_k = \overline{x} \vee \alpha$. La clause C_k peut alors être résolue avec $C_{m_1}, \ldots, C_{m_k}$ pour dériver $\overline{x} \vee \beta$. Ainsi, toute étape commune dans la résolution entre C'_i et C'_j ne sera pas dupliquée, ce qui permettra une *compression* de la preuve.

Cet argument peut être généralisé. Supposons que les clauses C_i et C_j diffèrent de plus de un littéral et qu'elles soient de la forme $C_i = \bar{l}_1 \vee \alpha \vee \beta$ et $C_j = \bar{l}_2 \vee \alpha \vee \gamma$, où $\alpha \vee \beta \vee \gamma$ ne soit pas une tautologie. Si il existe une séquence de clauses $C_{m_1}, \ldots, C_{m_k}$ telle que C_i et C_j entrent en résolution avec $C_{m_1}, \ldots, C_{m_k}$ pour dériver $C'_i = \bar{l}_1 \vee \delta$ et $C'_j = \bar{l}_1 \vee \delta$ où $\beta \subseteq \delta$ et $\gamma \subseteq \delta$, nous pouvons alors appliquer la résolution étendue pour éviter de dupliquer la résolution sur $C_{m_1}, \ldots, C_{m_k}$.

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. Ce point est primordial pour permettre l'introduction des variables en examinant la preuve au fur et à mesure de sa construction. La définition suivante formalise ce schéma comme un système de preuve par résolution.

Definition 1 (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, il est possible d'introduire la nouvelle variable $z \Leftrightarrow l_1 \vee l_2$ si il existe deux clauses $C_i \equiv \bar{l}_1 \vee \alpha$ et $C_j \equiv \bar{l}_2 \vee \beta$ avec $i < k, j < k$, où α et β sont des disjonctions vérifiant $l \in \alpha \implies \bar{l} \notin \beta$.*

3.2 Discussion sur le pouvoir de LER

En pratique, LER diffère de ER dans la mesure où elle impose la présence de clauses d'une forme donnée avant d'introduire de nouvelles variables. En tout généralité, certaines de ces clauses peuvent ne jamais être dérivées, empêchant du même coup l'introduction des variables les plus pertinentes. Cependant, il faut noter un avantage indéniable de LER sur ER. Lors de la construction de la preuve, l'introduction des nouvelles variables est moins imprévisible que celle de ER. Intuitivement, on voit combien chaque variable introduite est directement reliée à la preuve. Cela atténue la nature non déterministe de ce système de 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 polynômiales pour ER peuvent être transformées en des preuves polynômiales pour LER. C'est le cas par exemple pour la preuve ER de Stephen COOK sur le problème des pigeons [5], ce qui permet de montrer par la même occasion que LER est plus puissant que la résolution générale. D'un autre coté, il ne faut pas oublier qu'il peut être difficile de générer les deux clauses nécessaires à l'introduction d'une nouvelle variable. On peut donc s'attendre à ce que la preuve minimale pour LER soit plus grande que la preuve minimale pour ER pour certaines familles de problèmes. 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.

Dans la mesure où les solveurs CDCL ont été décrit comme des algorithmes basés sur la résolution [10], nous pensons avoir assez d'informations durant la recherche pour implémenter LER et étendre la formules avec des variables qui aideront à produire des preuves plus courtes. Ces idées sont exploitées dans la section ci-dessous au travers d'un nouveau type de solveur utilisant cette restriction de la résolution étendue.

4 Un Solveur CDCL basé sur les extensions locales

Le schéma expliqué dans la section précédente restreint le nombre de paires de littéraux candidats pour la résolution étendue. Malgré cela, le nombre de candidats potentiels est trop grand pour espérer étendre systématiquement la formule avec tous les candidats possibles.

Rappelons ici que notre but initial est de construire un solveur efficace sur les instances applicatives (industrielles). Il est donc primordial de mettre au point une heuristique efficace (coût de calcul réduit, nombre de candidats raisonnable) de sélection des paires candidates. Cette restriction, si importante soit-elle, est prioritaire. On peut s'attendre à ce qu'elle ne nous permette plus de tirer pleinement parti des bénéfices de LER, en ne permettant plus la résolution efficace de certaines instances. Nous n'avons d'autres choix que de contrôler de manière drastique le nombre de variables que nous souhaitons ajouter.

Typiquement, dans un solveur CDCL, la majeure partie du raisonnement a lieu durant l'analyse de conflit. Ceci inclut l'apprentissage (voir algorithme 1), mais aussi la mise à jour des scores heuristiques (variables et clauses)

et plus réçemment l'évaluation des stratégies de redémarrages. Comme c'est là que nous pouvons examiner facilement les clauses dérivées, c'est également un endroit logique pour détecter les paires de littéraux candidates à l'extension.

4.1 Détection rapide (mais incomplète) des paires de littéraux intéressants

La nouvelle restriction que nous imposons à notre solveur pour qu'il puisse rester efficace, en comparaison avec LER est la suivante. Nous concentrons notre effort sur les paires de clauses de la forme $\bar{l}_1 \vee \alpha, \bar{l}_2 \vee \alpha$. Encore une fois, cette nouvelle restriction peut réduire la puissance du système de preuves sous-jaçent à notre solveur. Elle va cependant permettre de conserver un nombre de candidats à l'extension relativement faible, ce qui, on l'espère, va s'avérer payant en pratique.

De plus, cette nouvelle restriction n'est pas sans lien avec les mécanismes d'apprentissages des solveurs CDCL. Ansi, intuitivement, les heuristiques des solveurs (notemment grâce à la sauvegarde de phases) leur permettent de rester dans le même espace de recherche malgré les redémarrages. Il semble donc naturel de penser que les paires de clauses correspondant au schéma $\bar{l}_1 \vee \alpha$, $\bar{l}_2 \vee \alpha$ soient produites de manière rapprochées durant la recherche. On peut donc penser réduire la fenêtre des clauses candidates à examiner pour détecter ces paires de clauses. Dans notre implantation, nous avons poussé ce raisonnement à l'extrême, en ne cherchant que les paires de clauses produites *successivement*. Ceci nous permet, de plus, de supposer que les littéraux $\bar{l}_1$ et $\bar{l}_2$ sont en fait les littéraux FUIP découverts durant l'analyse de conflit. Ainsi, notre processus de sélection des candidats est le suivant : si deux clauses successivement apprises par le solveur sont de la forme $\bar{l}_1 \vee \alpha$, $\bar{l}_2 \vee \alpha$, alors le solveur introduit la variable $z \Leftrightarrow l_1 \vee l_2$ (si la paire de littéraux l_1, l_2 n'a pas déjà étendue précédemment). Cette détection s'intègre hormonieusement au cadre classique des solveurs CDCL. Nous verrons dans la partie expérimentale que ce schéma de détection, même s'il semble très restrictif, se produit assez souvent durant la recherche.

Applications successives de la résolution étendue Si n clauses apprises successives sont de la forme $l_i \vee C$ $(1 \leq i \leq n)$ alors nous ajoutons la variable $z_1 \Leftrightarrow \bar{l}_1 \vee \bar{l}_2$, mais aussi $z_i \Leftrightarrow \bar{l}_i \vee \bar{l}_{i+1}$ pour $2 \leq i < n$. Il était également possible d'ajouter à la place les lemmes $z_i \Leftrightarrow \overline{z}_i \vee \bar{l}_{i+1}$ en lieu et place de $z_i \Leftrightarrow \bar{l}_i \vee \bar{l}_{i+1}$ mais les résultF106.c7tie exp (non6.c7tprésentés26648(ici))-.c7tont mont
la première approche était la plus efficace.

4.2 Injection des variables étendues dans les nouvelles clauses

À chaque fois qu'une nouvelle variable $z \Leftrightarrow l_1 \vee l_2$ est introduite, nous devons nous assurer que toutes les nouvelles 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 ieemple, si la sous-clause β est falsifiée, la clause $l_1 \vee l_2 \vee \beta$ n'est pas unitaire, mais la clause $z \vee \beta$ l'est. Les tests ie e-ment5 z n'apparaît pas dans l'analyse de conflit et ne participe donc pas à la preuve. Par soucis d'efficacité, nous ne réduisons pas les clauses apprises avant la création de la variable.

Afin de gérer efficacement cette étape de réduction, nous maintenons une table de hachage de chaque paire de littéraux (l_1, l_2). Celle-ci est sondée à chaque clause assertive générée afin de remplacer les paires par le littéral étendu associé. Notez que cela peut impliquer un choix. Supposons la clause $C \equiv l_1 \vee l_2 \vee l_3 \vee \beta$ et les variables étendues $z_1 \Leftrightarrow l_1 \vee l_2$ et $z_2 \Leftrightarrow l_2 \vee l_3$

redémarrage ni de remettre à jour les décision et les raisons qui en découlent.

4.4 Nettoyage des lemmes jugés inutiles

Afin d'optimiser les performances, les solveurs CDCL réduisent régulièrement la base des clauses apprises durant la recherche. Le même argument peut être appliqué aux variables introduites ainsi qu'aux clauses correspondantes. Nous avons donc opté pour supprimer les variables étendues ayant un petit score VSIDS, indiquant qu'elles ne sont pas très utiles à la preuve. Chaque fois que l'on effectue un nettoyage de la base de clauses, on supprime ainsi la moitié des variables étendues.

5 Évaluation expérimentale

Dans cette traditionnelle section expérimentale, nous proposons de comparer deux solveurs CDCL de référence, MINISAT et GLUCOSE, dans leurs versions initiales, avec leurs versions modifiées (intégrant la résolution étendue), respectivements appelées MINISATER et GLUCOSER. Nous avons tout de même ajouté à la comparaison PRECOSAT qui est, avec GLUCOSE, l'autre gagnant de la dernière compétition SAT. Notez que la version de MINISAT utilisée par la suite diffère quelque peu de la version disponible sur le web : nous avons changé la stratégie de redémarrage en utilisant une série de Luby (demarrant à 32) et nous avons également ajouté la sauvegarde de la phase. Ces modifications améliorent grandement les performances de MINISAT sur les problèmes industriels et sont maintenant adoptées par de nombreux solveurs.

Nous utilisons deux types de familles de benchmarks : (1) les instances de type application (industrielles) issues des compétitions SAT'07 et SAT'09 [16] et (2) des instances connues pour être dures pour les algorithmes basés sur la résolution (problème des pigeons, problèmes Urquhart [20]) ou expérimentalement connues pour être difficiles. Nous avons utilisé un cluster de quad-core Intel XEON X5550 – 2.66 GHz avec 32 go de RAM. Le temps limite est fixé à 5000 secondes et les résultats sont reportés en secondes.

5.1 Résultats obtenus sur les compétitions

La table 1 résume les résultats obtenus sur les instances industrielles des deux dernières compétitions. Il faut noter ici combien ces instances sont difficiles. Même une petite amélioration dans le nombre de problèmes résolus doit être vu comme un progrès non négligeable (notons que PRECOSAT et GLUCOSE, les deux vainqueurs de 2009, ont résolu le même nombre d'instances dans la catégorie SAT +UNSAT). De plus le réel challenge dans notre travail était de montrer que l'on pouvait augmenter la puissance de raisonnement des solveurs CDCL tout en gardant de bonnes performances générales.

	SAT 07	SAT 09
PRECOSAT	167 (91 - 76)	211 (129 - **82**)
GLUCOSE	185 (111 - 74)	211 (132 - 79)
GLUCOSER	**191** (**113** - **78**)	**213** (**133** - 80)
MINISAT	142 (81 - 61)	190 (118 - 72)
MINISATER	146 (85 - 61)	198 (123 - 75)

TAB. 1 – Performance de PRECOSAT et de MINISAT et GLUCOSE avec et sans LER. Les instances proviennent de la catégorie industrielle/applicaiton des compétitions 2007 (234 instances) et 2009 (292 instances). Pour chaque solveur, nous reportons le nombre d'instances résolues avec entre parenthèses, le nombre d'instances SAT et UNSAT.

5.2 Résultats obtenus sur des instances reconnues difficiles

La table 2 met en avant les resultats obtenus sur deux types d'instances problématiques. Les premières sont les instances connues pour être difficiles pour la résolution, comme celles codant le problème des pigeons et les instances Urquhart. Tout d'abord, notons que MINISATER n'améliore pas le solveur de base sur les instances des pigeons. En fait, MINISAT est clairement le meilleur solveur sur ce type d'instances. Nous pouvons néanmoins proposer une explication pour ces mauvais résultats, certes assez surprenant. Nous avons ainsi, par ailleurs, généré les clauses qui codent le problème des pigeons en y ajoutant les variables étendues utilisées par COOK pour faire une preuve courte [5]. Il a été assez surprenant de voir que ces instances ont également été difficiles pour MINISAT. Cela montre que l'utilisation des variables étendues n'est pas suffisante pour obtenir à coup sûr de bons résultats. Nous devons également modifier l'heuristique de branchement pour forcer ces variables à être utilisées durant le processus de résolution. Le second argument que nous pouvons apporter à ces mauvais résultats est que MINISATER et GLUCOSER améliorent de manière significative leur solveur de base lors de la résolution des instances qui encodent le problème des pigeons fonctionnel, admettant plus de contraintes. Malgré tout, même dans ce cas, les solveurs étendus n'ont pas un comportement polynômial par rapport à la taille des problèmes.

À l'opposé, les résultats sont clairement plus positifs sur les instances Urquhart. MINISATER et GLUCOSER améliorent grandement leurs solveurs de base même si, ici encore, leurs résultats ne semblent pas évoluer polynomialement avec la taille des instances.

Enfin, la partie la plus importante de la table 2 est en rapport avec un ensemble d'instances connues expé-

rimentalement comme étant difficiles. Certaines d'entre elles n'ont pas été résolues durant la dernière compétition. Cela montre clairement que GLUCOSER, et dans une moindre mesure, MINISATER, résolvent des instances innaccessibles à leur solveur de base. Avant de conclure, voici quelques points qui peuvent être soulignés. Tout d'abord, les solveurs introduisent de nombreuses nouvelles variables, en gros, une tous les 1000 conflits. De plus, et cela n'est pas rapporté sur les tables, les solveurs branchent fréquemment sur les nouvelles variables. Ce qui implique qu'elles ont un score VSIDS elevé, et donc qu'elles sont utilisées durant le processus de résolution. Cela montre que GLUCOSER et MINISATER construisent une preuve LER, et pas seulement une preuve de résolution. Ainsi, malgré la restriction apportée à LER nous arrivons à introduire de nouvelles variables et LER admet des preuves beaucoup plus courtes sur ces instances que les solveurs CDCL classiques.

6 Conclusions

Dans cet article, nous avons proposé une restriction de la résolution étendue présentant deux avantages non négligeables par rapport à la résolution étendue classique. Tout d'abord, les variables introduites sont en relation directe avec la preuve construite. De plus, le nombre de paires de littéraux candidates à l'extension est fortement réduit. Ces deux avantages simplifient la création d'un solveur CDCL basé sur ce système de preuves. L'implantation résultante sur deux solveurs différents, y compris un solveur état de l'art, en a amélioré leurs performances sur des instances difficiles issues des dernières compétitions.

Ce travail laisse de nombreuses questions ouvertes. D'un point de vue pratique nous souhaitons mettre au point des heuristiques permettant d'introduire et de supprimer beaucoup plus fréquemment des variables, ceci, sans surcoût important. On peut par exemple imaginer tenir compte d'un ensemble de nogoods à comparer plutôt que de se limiter aux deux derniers. D'un point de vue théorique, nous souhaitons de plus déterminer la place exacte de LER dans la hiérarchie des systèmes de preuves. Si l'on arrive à prouver que ce dernier p-simule ER, cela pourrait être une étape importante pour comprendre la puissance de ER et aider à son exploitation dans les solveurs SAT.

Remerciements

Les auteurs tiennent à remercier Nicolas Prcovic pour avoir suggéré l'utilisation des instances de pigeons « préétendues ».

Références

[1] R. J. Bayardo and R. C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 203–207, Providence, RI, 1997.

[2] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22 :319–351, 2004.

[3] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *J. ACM*, 48(2) :149–169, 2001.

[4] Philippe Chatalic and Laurent Simon. Multiresolution for SAT checking. *International Journal on Artificial Intelligence Tools*, 10(4) :451–481, 2001.

[5] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4) :28–32, 1976.

[6] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5 :394–397, 1962.

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

[8] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39 :297–308, 1985.

[9] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively p-simulate general propositional resolution. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 283–290, 2008.

[10] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 2318–2323, 2007.

[11] J. P. Marques Silva and K. A. Sakallah. GRASP—a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5) :506–521, May 1999.

[12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference*, pages 530–535, 2001.

[13] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, volume 5732, chapter 51, pages 654–668. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

benchmark	SAT ?	PRECOSAT	GLUCOSE	GLUCOSER	MINISAT	MINISATER
hole-11	UNSAT	90s / 1,314	78s / 552	127s / 848 / 930	7s / 148	20s / 359 / 2195
hole-12	UNSAT	963s / 8,112	1,167s / 4,588	208s / 1,333 / 1,334	40s / 405	363s / 1,825 / 6,927
Urq3_5	UNSAT	142s / 4,570	21s / 791	8s / 235 / 1,478	398s / 2,119	10s / 318 / 2,702
Urq4_5	UNSAT	–	3,106s / 28,181	26s / 527 / 3,103	–	11s / 333 / 3,191
Urq5_5	UNSAT	–	–	545s / 5,021 / 7,298	–	107s / 1,488 / 7,503
aloul-chnl11-13	UNSAT	–	–	310s / 1,794 / 1,233	130s / 808	88s / 859 / 4,665
SAT_dat.k75†	UNSAT	–	–	614s / 5,544 / 23,042	–	–
dated-5-19-u†	UNSAT	–	–	3,127s / 9,063 / 9,202	–	–
9dlx_vliw_at_b_iq8†	UNSAT	–	4,322s / 6,672	3,609s / 4,733 / 14,490	–	–
sortnet-8-ipc5 -h19-sat†	SAT	3,395s / 2,179	–	3,672s / 7,147 / 6,921	–	–
vmpc_34	SAT	–	2,616s / 15,833	1,565s / 10,022 / 31,576	–	–
rbcl_xits_08	UNSAT	1,286s / 3,600	–	3,067s / 8,069 / 8,916	–	–
simon-s02b- k2f-gr-rcs-w8	UNSAT	–	–	3,622s / 7,904 / 13,567	–	3,797s / 5,121 / 30,890

TAB. 2 – Résultats sur une sélection d'instances difficiles. Pour chaque instance, nous reportons, si elle est satisfaisable, le temps nécessaire par chaque solveur pour la résoudre, le nombre de conflits (en milliers) et le nombre de variables étendues (après le "/", si résolution étendue il y a eu) ou "–" si le solveur ne résoud pas l'instance dans le temps imparti. Les instances annotées par †n'ont pas été résolues aux compétitions SAT'07 et SAT'09. Les autres ont été résolues par au plus 5 solveurs.

[14] 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.

[15] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, 1965.

[16] `http://www.satcompetition.org`, 2009.

[17] Bas Schaafsma, Marijn Heule, and Hans van Maaren. Dynamic symmetry breaking by simulating zykov contraction. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584, chapter 22, pages 223–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[18] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[19] G. Tseitin. On the complexity of proofs in propositional logics. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning : Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

[20] A. Urquhart. Hard examples for resolution. *JACM*, 34(1) :209–219, 1987.

[21] Alasdair Urquhart. The complexity of propositional proofs. pages 332–342, 2001.

[22] Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science, chapter 40, pages 580–594. 2005.

[23] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of IEEE/ACM International Conference on Computer Design (ICCAD)*, pages 279–285, 2001.