Combinaison des règles d'inférence et de la sous-estimation des bornes inférieures pour Max-SAT

Chu Min Li, Felip Manya, Nouredine Ould Mohamedou, Jordi Planes

Combinaison des Règles d'Inférence et de la sous-Estimation des bornes inférieures pour Max-SAT

Chu Min LI, Felip Manyà, Nouredine Ould Mohamedou et Jordi Planes

MIS, Université de Picardie Jules Verne
5 Rue du Moulin Neuf 80000 Amiens, France.
IIIA-CSIC, Campus UAB
08193 Bellaterra Spain.
University of Southampton
Southampton SO17 1BJ United Kingdom.
{chu-min.li, nouredine.ould}@u-picardie.fr,felip@iiia.csic.es,jp3@ecs.soton.ac.uk

Résumé

Le calcul de la borne inférieure (LB) pour les solveurs Max-SAT basés sur la méthode de Séparation et Évaluation (Branch and bound) a généralement deux composantes : (i) la *composante sous-estimation*, qui détecte les sous-formules disjointement inconsistantes et considère le nombre de ces sous-formules détectées comme une *sous-estimation* de LB, et (ii) la *composante inférence*, qui applique les règles d'inférence qui, dans le meilleur des cas, explicitent une contradiction permettant d'incrémenter LB. Dans ce papier, nous nous concentrons sur l'interaction entre la *composante sous-estimation* et la règle d'inférence dite *résolution de cycle*, qui a été récemment montrée très puissante dans deux solveurs Max-SAT de l'état-de-l'art : MAX-DPLL et MaxSatz. MAX-DPLL applique la règle de résolution de cycle de façon exhaustive, mais ne combine pas son application à la composante calculant la sous-estimation, tandis que MaxSatz le fait mais avec une application limitée de la résolution de cycle. Notre objectif est d'obtenir une meilleure intégration de la règle de *résolution de cycle* avec la *composante sous-estimation* afin d'améliorer la qualité de LB. Dans ce but, nous avons défini plusieurs heuristiques qui guident l'application de la résolution de cycle dans MaxSatz. La plus performante de ces heuristiques nous a permis d'obtenir une variante de MaxSatz, appelée $MaxSatz_c$, qui applique la résolution de cycle lorsque la *composante sous-estimation* est susceptible de détecter des sous-formules inconsistantes. Notre étude expérimentale montre que $MaxSatz_c$ améliore substantiellement MaxSatz sur un nombre considérable d'instances difficiles.

Abstract

The lower bound (LB) computation method of modern branch and bound Max-SAT solvers has generally two components : (i) the underestimation component, which detects disjoint inconsistent subformulas and takes the number of detected inconsistent subformulas as an underestimation of the LB, and (ii) the inference component, which applies inference rules that, in the best case, makes explicit a contradiction which allows us to increment the LB. In this paper we focus on the interaction of the underestimation component with the cycle resolution inference rule, which recently has proven very powerful in two state-of-the-art Max-SAT solvers : MAX-DPLL and MaxSatz. MAX-DPLL applies cycle resolution exhaustively but does not combine its application with the underestimation component, while MaxSatz combines the underestimation component with a limited application of cycle resolution. Our aim is to obtain a better integration of the cycle resolution rule with the underestimation component to improve the quality of the LB. To this end, we have defined several heuristics which guide the application of cycle resolution in MaxSatz. The best performing heuristic allows us to obtain a variant of MaxSatz, called $MaxSatz_c$, which applies cycle resolution when the underestimation component is likely to detect an inconsistent subformula. Our experimental investigation shows that $MaxSatz_c$ substantially improves MaxSatz on a considerable number of hard instances.

1 Introduction

Les solveurs exacts Max-SAT les plus compétitifs (par exemple ([1] ; [5] ; [8] ; [11] ; [12] ; [13])) implémentent diverses variantes de la méthode Séparation Et Évaluation (BnB) selon le schéma suivant : Pour une formule CNF donnée, BnB explore l'espace de recherche de toutes les affectations possibles dans une recherche en *profondeur d'abord*. A chaque noeud, BnB compare la borne supé-

rieure (UB), qui est la meilleure solution déjà trouvée (à ce moment là) pour une affectation complète, avec la borne inférieure (LB), qui est la somme du nombre de clauses falsifiées par l'affectation partielle courante, plus, une sous-estimation du nombre de clauses qui seraient falsifiées si cette affectation partielle était complétée (achevée). Si $LB \geq UB$, BnB coupe le sous-arbre en-dessous du noeud courant et retourne en arrière (i.e. en faisant un backtrack). Si $LB < UB$, BnB tente de trouver une meilleure solution en instanciant une nouvelle variable en plus. La solution de Max-SAT est la valeur de UB après exploration de la totalité de l'espace de recherche.

La méthode de calcul de LB implémentée dans les solveurs MaxSatz [11] et MiniMaxSat [6] qui sont basés sur BnB (et qui ont été les plus performants solveurs de *Max-SAT Evaluation 2007*) a deux composantes : (i) la composante de sous-estimation, qui détecte les sous-formules disjointement inconsistantes et considère le nombre de ces sous-formules détectées comme une sous-estimation, et (ii) la composante d'inférence, qui applique les règles d'inférence qui, dans le meilleur des cas, explicitent une contradiction (clause vide) qui permet d'incrémenter LB. Les deux composantes sont appliquées, en principe, à chaque noeud de l'espace de recherche.

La composante de la sous-estimation utilise la propagation unitaire [9] (ou la propagation unitaire renforcée avec la détection de littéral d'échec [10]) pour détecter les sous-formules inconsistantes avec un faible coût car la propagation unitaire peut être implémentée de manière très efficace. Elle a l'inconvénient de calculer la sous-estimation en local, au noeud courant, et par conséquent, une nouvelle sous-estimation doit être recalculée à partir de zéro à chaque noeud, même lorsque des contradictions sont retrouvées dans l'espace de recherche en-dessous du noeud courant. La composante inférence surmonte ce problème car toute clause vide dérivée du noeud courant par l'application de règles d'inférence reste dans les noeuds descendants. En outre, les règles d'inférence ajoutent des nouvelles clauses qui peuvent aider à détecter des nouvelles sous-formules inconsistantes ou à appliquer une règle d'inférence supplémentaire. Leur inconvénient est qu'elles augmentent le coût en temps d'exécution, car les clauses ajoutées doivent être retirées lors des *backtracking*.

Dans ce papier, on étudie l'interaction entre la *composante sous-estimation* et celle de l'*inférence*. Et ceci, quand il y a une structure particulière (que nous appelons la *structure de cycle*) dans une instance Max-SAT. Nous élaborons plusieurs stratégies visant à améliorer la qualité de LB en combinant la composante de la sous-estimation et celle de la règle d'inférence qui traite cette *structure de cycle*. De ce fait, on obtient une variante de *MaxSatz*, appelée $MaxSatz_c$, qui incorpore notre plus performante stratégie. Notre étude expérimentale (avec des instances relevant de *Max-SAT Evaluation 2007* et d'autres de Max-2SAT, Max-3SAT et MaxCUT) montre que $MaxSatz_c$ est nettement plus performant que *MaxSatz*.

2 Préliminaires

En logique propositionnelle, une variable x_i peut prendre les valeurs 0 (pour Faux) ou 1 (pour Vrai). Un littéral l_i est une variable x_i ou sa négation $\overline{x_i}$. Une clause est une disjonction de littéraux et une formule CNF est un ensemble de clauses (ou un ET logique de ces clauses). Une affectation de valeurs de vérité aux variables propositionnelles satisfait le littéral x_i si x_i prend la valeur 1 et satisfait le littéral $\overline{x_i}$ si x_i prend la valeur 0, satisfait une clause si elle satisfait au moins un de ses littéraux et satisfait une formule CNF si elle satisfait toutes les clauses de cette formule. Une clause vide, notée $\square$, ne contient aucun littéral et ne peut donc être satisfaite. Un littéral l, dans une formule CNF ϕ, est un *littéral d'échec* si la propagation unitaire détecte une contradiction dans $\phi \wedge l$ mais pas dans ϕ.

Max-SAT est, pour une formule CNF, le problème qui consiste à trouver une affectation de valeurs de vérité aux variables propositionnelles, qui minimise le nombre de clauses falsifiées[1].

Les formules Max-SAT ϕ_1 et ϕ_2 sont équivalentes si ϕ_1 et ϕ_2 ont le même nombre de clauses falsifiées pour toute affectation complète de toutes les variables de ϕ_1 et ϕ_2. Une règle d'inférence Max-SAT est correcte si elle transforme une instance en une autre qui lui est équivalente.

3 Travaux précédents

La *Composante sous-Estimation :* La plus simple sous-estimation [14] consiste à compter le nombre de sous-formules disjointement inconsistantes, formées par une clause unitaire avec le littéral l et une autre clause unitaire avec le littéral complémentaire de l ($\overline{l}$). Les sous-estimations considérant des clauses plus longues incluent la *règle-star* et UP. En *règle-star* [1], la sous-estimation de LB est le nombre de sous-formules disjointement inconsistantes de la forme $\{l_1,...,l_k,\overline{l}_1 \vee ... \vee \overline{l}_k\}$ dans l'instance.

En UP [9], la sous-estimation est le nombre de sous-formules disjointement inconsistantes qui peuvent être détectées avec une propagation unitaire (UP). UP fonctionne de la façon suivante : Elle applique la propagation unitaire jusqu'à ce qu'une contradiction apparaisse. Ensuite, UP identifie, en inspectant le graphe d'implication, un sous-ensemble de clauses à partir duquel une réfutation unitaire peut être construite, et tente d'en déduire de nouvelles contradictions parmi les autres clauses avec une nouvelle propagation unitaire. L'ordre dans lequel les clauses unitaires sont propagées a un impact évident sur la qualité de

[1] Ce qui revient à vouloir maximiser le nombre de clauses satisfaites.

LB [10]. Récemment, une version de l'UP dans laquelle le calcul de LB est rendue plus incrémental, a été mise au point dans [4].

UP peut être améliorée par la détection des littéraux d'échec (UP_{FL}) comme suit [10] : Étant donnée une instance Max-SAT ϕ sur laquelle nous avons déjà appliqué UP et une variable x apparaissant, à la fois, positivement et négativement dans ϕ, nous appliquons UP à $\phi \wedge x$ et à $\phi \wedge \bar{x}$. Si UP génère une contradiction, à la fois, à partir de $\phi \wedge x$ et $\phi \wedge \bar{x}$, alors l'union des deux sous-ensembles inconsistants identifiés par UP, en excluant x et $\bar{x}$, est un sous-ensemble inconsistant de ϕ. Puisque l'application de la détection de littéraux d'échec, à chaque variable, est coûteux en temps, la détection est appliquée à un nombre réduit de variables en pratique : Soit $Var(\phi)$ l'ensemble des variables propositionnelles apparaissant dans ϕ tel que chacune de ces variables (i) n'apparaît pas dans les clauses unitaires et (ii) apparaît au moins deux fois positivement et deux fois négativement dans des clauses binaires. UP_{FL} détecte, pour chaque variable x de $Var(\phi)$, si x et $\bar{x}$ sont deux littéraux d'échec de ϕ. Une fois qu'une sous-formule inconsistante γ est détectée, γ est supprimée de ϕ, la sous-estimation est incrémentée de 1 et le sous-ensemble de variables $Var(\phi)$ dans lequel des littéraux d'échec sont recherchés est mis à jour, en tenant compte de la nouvelle formule CNF dérivée. Les variables apparaissant dans des clauses unitaires ne sont pas considérées parce qu'elles ne conduiront pas à une contradiction si UP a été appliquée à ϕ. Le fait de sélectionner des variables avec au moins deux occurrences positives et deux négatives dans des clauses binaires, est déterminé empiriquement. Ces variables donnent au moins deux nouvelles clauses unitaires lorsqu'une valeur de vérité leur est affectée.

La *Composante Inférence :* Une alternative pour améliorer la qualité de LB consiste à appliquer des règles d'inférence efficaces pour Max-SAT. Malheureusement, la règle de résolution appliquée en SAT (c'est-à-dire, déduire $D \vee D$' à partir de $x \vee D$ et $\bar{x} \vee D$') préserve la satisfiabilité des formules mais pas leur équivalence, et ne peut donc être appliquée aux instances Max-SAT.

Des règles d'inférence efficaces pour Max-SAT, analogues à celles appliquées en SAT incluent la *règle de littéral pur* et celle de la *clause presque commune* [2] (si une instance Max-SAT contient une clause $x \vee D$ et une clause $\bar{x} \vee D$, où D est une disjonction de littéraux ; les deux clauses sont remplacées par D. En pratique, cette règle est appliquée lorsque D contient au plus un littéral).

Une règle d'inférence qui ajoute des clauses supplémentaires est la *règle-star* suivante [11] : Si $\phi_1 = \{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\} \cup \phi'$, alors $\phi_2 = \{\square, l_1 \vee l_2\} \cup \phi'$ est équivalente à ϕ_1. Cette règle peut aussi être présentée comme suit :

$$\begin{array}{c} l_1 \\ \bar{l}_1 \vee \bar{l}_2 \\ l_2 \end{array} \quad \Longrightarrow \quad \begin{array}{c} \square \\ l_1 \vee l_2 \end{array} \qquad (1)$$

Notons que la règle détecte une contradiction à partir de l_1, $\bar{l}_1 \vee \bar{l}_2$, l_2 et par conséquent, remplace ces clauses par une clause vide. De plus, elle ajoute la clause $l_1 \vee l_2$ pour s'assurer que le nombre de clauses falsifiées de ϕ_1 et de ϕ_2 reste le même quand l_1= 0, l_2= 0 (dans ce cas il y a 2 clauses falsifiées). En plus des règles précédentes, MaxSatz [11] incorpore les règles suivantes :

$$\begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_2 \vee l_3 \\ \ldots \\ \bar{l}_k \vee l_{k+1} \\ \bar{l}_{k+1} \end{array} \quad \Longrightarrow \quad \begin{array}{c} \square \\ l_1 \vee \bar{l}_2 \\ l_2 \vee \bar{l}_3 \\ \ldots \\ l_k \vee \bar{l}_{k+1} \end{array} \qquad (2)$$

$$\begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_1 \vee l_3 \\ \bar{l}_2 \vee \bar{l}_3 \end{array} \quad \Longrightarrow \quad \begin{array}{c} \square \\ l_1 \vee \bar{l}_2 \vee \bar{l}_3 \\ \bar{l}_1 \vee l_2 \vee l_3 \end{array} \qquad (3)$$

$$\begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_2 \vee l_3 \\ \ldots \\ \bar{l}_k \vee l_{k+1} \\ \bar{l}_{k+1} \vee l_{k+2} \\ \bar{l}_{k+1} \vee l_{k+3} \\ \bar{l}_{k+2} \vee \bar{l}_{k+3} \end{array} \quad \Longrightarrow \quad \begin{array}{c} \square \\ l_1 \vee \bar{l}_2 \\ l_2 \vee l_3 \\ \ldots \\ l_k \vee \bar{l}_{k+1} \\ l_{k+1} \vee l_{k+2} \vee \bar{l}_{k+3} \\ \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3} \end{array} \qquad (4)$$

Indépendamment et en parallèle aux règles de MaxSatz, plusieurs règles pour Max-SAT pondéré, y compris la *résolution de chaîne* (qui est équivalente à la règle 2 dans le cas non pondéré) et la résolution de cycle, ont été définies dans [5, 8]. Ces règles ont été incorporées dans le solveur MAX-DPLL [8]. MAX-DPLL implémente une résolution de cycle limitée à 3 variables :

$$\begin{array}{c} \bar{l}_1 \vee l_2 \\ \bar{l}_1 \vee l_3 \\ \bar{l}_2 \vee \bar{l}_3 \end{array} \quad \Longrightarrow \quad \begin{array}{c} \bar{l}_1 \\ l_1 \vee \bar{l}_2 \vee \bar{l}_3 \\ \bar{l}_1 \vee l_2 \vee l_3 \end{array} \qquad (5)$$

Dans la suite, quand nous disons "*résolution de cycle*", nous entendons la résolution de cycle restreinte à 3 variables. Il est à noter que les règles 3 et 4 dans MaxSatz et la résolution de cycle dans MAX-DPLL, traitent une structure spéciale : $\bar{l}_1 \vee l_2, \bar{l}_1 \vee l_3, \bar{l}_2 \vee \bar{l}_3$ que nous appelons *structure de cycle*. Notons qu'une structure de cycle ne constitue pas, à elle seule, une sous-formule inconsistante. MAX-DPLL applique la résolution de cycle de manière illimitée : il remplace tout sous-ensemble de trois clauses binaires représentant une structure de cycle par une clause unitaire et deux

clauses ternaires. Quant à MaxSatz, il applique la résolution de cycle de façon limitée : il l'applique uniquement lorsque la propagation unitaire détecte une contradiction contenant une structure de cycle.

La résolution de cycle est très utile quand son application nous permet de déduire une sous-formule inconsistante car la LB est incrémentée dans un tel cas. En revanche, il n'est pas clair si les clauses déduites (une unitaire et deux ternaires) seront plus utiles que les trois clauses binaires d'origine. Dans le reste de ce papier, nous étudions l'utilité de la résolution de cycle lorsqu'aucune sous-formule inconsistante n'est détectée par une application de résolution de cycle.

Les règles précédentes peuvent être considérées comme des applications des règles de résolution complète pour Max-SAT, définies dans [7, 3].

Combinaison de la sous-Estimation à l'Inférence : Dans les récents et performants solveurs, les composantes *sous-estimation* et *inférence* coopèrent plutôt au lieu de travailler de façon indépendante. Par exemple, MaxSatz utilise la propagation unitaire pour détecter des sous-formules inconsistantes et, dès qu'une contradiction est trouvée, il applique une règle d'inférence si la sous-formule inconsistante détectée correspond aux prémisses d'une de ses règles d'inférence. Ceci est effectué par l'inspection du graphe d'implication construit par la propagation unitaire. Remarquons que les règles 3 et 4 appliquent la résolution de cycle en utilisant la propagation unitaire.

On note que les trois clauses binaires $\bar{l}_1 \vee l_2, \bar{l}_1 \vee l_3, \bar{l}_2 \vee \bar{l}_3$ dans une *structure de cycle* impliquent le littéral d'échec l_1. Cependant, quand les versions existantes de MaxSatz, enrichies d'une détection de littéral d'échec dans la composante sous-estimation, détectent deux littéraux complémentaires d'échec, ils incrémentent la sous-estimation mais ne vérifient pas si la résolution de cycle peut être appliquée. Donc, une combinaison de résolution de cycle à la détection du littéral d'échec pourrait être un moyen naturel pour améliorer MaxSatz.

MiniMaxSat utilise la propagation unitaire pour détecter des sous-formules inconsistantes et, une fois une contradiction est trouvée, il applique la règle de résolution de Max-SAT à la sous-formule inconsistante détectée si la plus grande résolvante dans la réfutation a une arité inférieur à 4, sinon il incrémente tout simplement la sous-estimation.

4 La résolution de cycle dans MaxSatz

Dans les exemples suivants, nous montrerons l'intérêt d'appliquer la résolution de cycle dans différents scénarios qui ne sont pas pris en compte dans la dernière version publiée de MaxSatz[2], appelée MaxSatz-07 dans ce papier. MaxSatz-07 comprend la détection de littéraux d'échec dans la *composante sous-estimation* et est la version utilisée dans *Max-SAT Evaluation 2007*.

Supposons que MaxSatz-07 doive résoudre une instance contenant les clauses (la sous-formule) $x_1 \vee x_2, x_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3, \bar{x}_1 \vee x_4, \bar{x}_1 \vee x_5, \bar{x}_4 \vee \bar{x}_5$.

La détection de littéral d'échec détecte la précédente sous-formule inconsistante, mais deux applications de résolution de cycle, qui ne sont pas exécutées dans MaxSatz-07, remplacent ces six clauses par $\square$, $\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$, $x_1 \vee x_2 \vee x_3$, $x_1 \vee \bar{x}_4 \vee \bar{x}_5$, $\bar{x}_1 \vee x_4 \vee x_5$. Donc, outre le fait d'expliciter une contradiction qui n'a pas besoin d'être redétectée dans le sous-espace au-dessous du noeud courant, les nouvelles clauses sont ajoutées et peuvent aider à détecter de nouvelles contradictions.

Regardons un autre exemple qui met en évidence l'utilité des clauses ajoutées par la résolution de cycle au cours de la détection de littéraux d'échec.
Supposons que l'instance contienne
$x_1 \vee x_{12}$, $x_1 \vee x_2$, $\bar{x}_2 \vee x_3$, $\bar{x}_2 \vee x_4$, $\bar{x}_3 \vee \bar{x}_4$,
$\bar{x}_1 \vee x_5$, $\bar{x}_5 \vee x_6$, $\bar{x}_1 \vee x_7$, $\bar{x}_6 \vee \bar{x}_7$,
$x_8 \vee \bar{x}_2$, $x_8 \vee x_3$, $x_8 \vee x_4$, $\bar{x}_8 \vee x_9$, $\bar{x}_8 \vee x_{10}$, $\bar{x}_8 \vee x_{11}$, $\bar{x}_9 \vee \bar{x}_{10} \vee \bar{x}_{11}$
La Règle 3 et la Règle 4 ne sont pas appliquées car il n'y a pas de clauses unitaires. Dans ce cas, la détection de littéral d'échec sur la variable x_1 donne, à partir de deux premières lignes, la sous-formule inconsistante
$x_1 \vee x_2$, $\bar{x}_2 \vee x_3$, $\bar{x}_2 \vee x_4$, $\bar{x}_3 \vee \bar{x}_4$, $\bar{x}_1 \vee x_5$, $\bar{x}_5 \vee x_6$, $\bar{x}_1 \vee x_7$, $\bar{x}_6 \vee \bar{x}_7$
Après suppression de cette sous-formule, la détection de littéral d'échec ne peut pas redétecter de nouvelles sous-formules inconsistantes. En revanche, si la résolution de cycle est appliquée à $\bar{x}_2 \vee x_3$, $\bar{x}_2 \vee x_4$, $\bar{x}_3 \vee \bar{x}_4$, ces clauses sont remplacées par $\bar{x}_2$, $x_2 \vee \bar{x}_3 \vee \bar{x}_4$, $\bar{x}_2 \vee x_3 \vee x_4$, puis la *composante sous-estimation* détecte alors deux sous-formules inconsistantes au lieu d'une seule : la première par la détection de littéral d'échec sur la variable x_1 :
$\bar{x}_2$, $x_1 \vee x_2$, $\bar{x}_1 \vee x_5$, $\bar{x}_5 \vee x_6$, $\bar{x}_1 \vee x_7$, $\bar{x}_6 \vee \bar{x}_7$
et la deuxième par la détection de littéral d'échec sur la variable x_8 :
$x_8 \vee \bar{x}_2$, $x_8 \vee x_3$, $x_8 \vee x_4$, $\bar{x}_8 \vee x_9$, $\bar{x}_8 \vee x_{10}$, $\bar{x}_8 \vee x_{11}$, $\bar{x}_9 \vee \bar{x}_{10} \vee \bar{x}_{11}$, $x_2 \vee \bar{x}_3 \vee \bar{x}_4$.

Notons que le littéral d'échec détecté en utilisant la propagation unitaire (x_1 dans l'exemple ci-dessus) n'a pas besoin d'apparaître dans la structure de cycle détectée ($\bar{x}_2 \vee x_3$, $\bar{x}_2 \vee x_4$, $\bar{x}_3 \vee \bar{x}_4$). Une fois un littéral d'échec détecté, la détection de la structure de cycle dans le graphe d'implication a un coût négligeable.

De manière générale, lorsque UP($\phi \wedge l$) et UP($\phi \wedge \bar{l}$) déduisent, à la fois, une contradiction, on nomme S_l l'ensemble de clauses ayant permis de déduire la contradiction dans UP($\phi \wedge l$). Si S_l contient la structure de cycle $\bar{l}_1 \vee l_2$, $\bar{l}_1 \vee l_3$, $\bar{l}_2 \vee \bar{l}_3$, alors l_1 a été instancié à *vrai* dans la propagation unitaire. Si la résolution de cycle est appliquée, les

[2] http ://www.laria.u-picardie.fr/~cli/EnglishPage.html

trois clauses binaires sont remplacées par $\bar{l}_1$ et deux clauses ternaires. S_l demeure inconsistant après suppression des trois clauses binaires et ajout de $\bar{l}_1$. Donc $(S_l \cup S_{\bar{l}}) - \{l, \bar{l}\}$ devient une sous-formule inconsistante plus petite. Les deux clauses ternaires obtenues par application de la résolution de cycle peuvent alors être utilisées pour détecter d'autres littéraux d'échec, ce qui augmente la probabilité d'obtenir de meilleures bornes inférieures. Dans l'exemple ci-dessus, $S_{\bar{x}_1} = \{\bar{x}_1, x_1 \vee x_2, \bar{x}_2 \vee x_3, \bar{x}_2 \vee x_4, \bar{x}_3 \vee \bar{x}_4\}$ demeure inconsistant après avoir remplacé $\bar{x}_2 \vee x_3$, $\bar{x}_2 \vee x_4$, $\bar{x}_3 \vee \bar{x}_4$ par $\bar{x}_2$ ($S_{\bar{x}_1}$ devient $\{\bar{x}_1, x_1 \vee x_2, \bar{x}_2\}$). Ainsi, après avoir incrémenté la sous-estimation de 1, nous avons deux autres clauses ternaires supplémentaires ($x_2 \vee \bar{x}_3 \vee \bar{x}_4$ et $\bar{x}_2 \vee x_3 \vee x_4$) qui peuvent être utilisées pour détecter de nouvelles contradictions. Si UP($\phi \wedge l$) déduit une contradiction, mais UP($\phi \wedge \bar{l}$) ne le fait pas, la sous-estimation ne peut être incrémentée. Dans ce cas, l'utilité d'appliquer la résolution de cycle, dans S_l, n'est pas claire.

Algorithm 1: *flAndCycle*(ϕ, l), fonction générique pour combiner la résolution de cycle et la détection de littéral d'échec

Input: Une instance Max-SAT ϕ et un littéral l
Output: ϕ , dans laquelle la résolution de cycle est peut-être appliquée, et une *sousEstimation*

```
1  begin
2      sousEstimation ⟵ 0 ;
3      if UP(φ ∧ {l}) génère une contradiction then
4          appliquer la résolution de cycle dans φ si S_l
           contient une structure de cycle ;
5          if UP(φ ∧ {l̄}) génère une contradiction then
6              appliquer la résolution de cycle dans φ si
               S_l̄ contient une structure de cycle ;
7              sousEstimation ⟵ 1 ;
8      return la nouvelle φ et sousEstimation ;
9  end
```

5 Différentes stratégies pour combiner la résolution de cycle et la détection de littéraux d'échec

L'algorithme 1 montre de manière générique une combinaison de résolution de cycle à la détection de littéraux d'échec. Notons premièrement que, si l n'est pas un littéral d'échec, $\bar{l}$ n'est pas détecté et aucune résolution de cycle n'est appliquée à aucune structure de cycle contenant l. En revanche, puisque l est détecté en premier lieu, la résolution de cycle peut être appliquée à une structure de cycle contenant $\bar{l}$, même si $\bar{l}$ n'est pas un littéral d'échec. Donc, l'ordre de traitement de l et $\bar{l}$ est important. Deuxièmement, la résolution de cycle est appliquée au plus une fois pour chaque littéral d'échec. Par exemple, si ϕ contient $\bar{x}_1 \vee x_2$, $\bar{x}_1 \vee x_3$, $\bar{x}_2 \vee \bar{x}_3$, $\bar{x}_1 \vee x_4$, $\bar{x}_1 \vee x_5$, $\bar{x}_4 \vee \bar{x}_5$ et x_1 est détecté comme un littéral d'échec, la résolution de cycle est uniquement appliquée à la structure de cycle détectée, par exemple : $\bar{x}_1 \vee x_2$, $\bar{x}_1 \vee x_3$, $\bar{x}_2 \vee \bar{x}_3$.

Dans le but d'analyser l'impact de la résolution de cycle sur la performance de MaxSatz, nous avons défini un certain nombre de solveurs qui appliquent, à différents "degrés", la résolution de cycle :

- **MaxSatz-07** : Version standard de MaxSatz. Outre la UP, elle implémente toutes les règles d'inférence de MaxSatz et la détection de littéral d'échec, dans la composante sous-estimation .
- **MaxSatz** : Version optimisée de MaxSatz-07. Les solveurs ci-dessous ont été implémentés à partir d'elle comme version de base.
- **MaxSatz$_c$** : Soit *occ2(l)* le nombre de clauses binaires contenant le littéral l dans ϕ. Si $occ2(l) > occ2(\bar{l}) \geq 2$, appeler *flAndCycle* (ϕ, l), sinon, si $occ2(\bar{l}) \geq occ2(l) \geq 2$, appeler *flAndCycle* (ϕ, $\bar{l}$). Notons que si *occ2(l)* > *occ2* ($\bar{l}$), la probabilité de trouver une contradiction quand l est vrai est plus petite que lorsque $\bar{l}$ est vrai. Ainsi, la résolution de cycle est appliquée probablement en cas où $\bar{l}$ et l sont tous les deux des littéraux d'échec. Comparé à MaxSatz, après détection de littéraux d'échec l et $\bar{l}$ et après incrémentation de la sous-estimation de 1, MaxSatz$_c$ a plus de clauses pour détecter d'autres littéraux d'échec en appliquant la résolution de cycle.
- **MaxSatz$_{c+}$** : Similaire à MaxSatz$_c$, mais si $occ2(l) > occ2(\bar{l}) \geq 2$, appeler *flAndCycle* ($\phi$, $\bar{l}$), sinon, si $occ2(\bar{l}) \geq occ2(l) \geq 2$, appeler *flAndCycle* ($\phi$, l). On remarque que, généralement, il y aura plus d'applications de résolution de cycle dans MaxSatz$_{c+}$ que dans MaxSatz$_c$. Ces sont les cas où la résolution de cycle est appliquée, mais la sous-estimation ne peut être incrémentée (i.e., le cas où l est littéral d'échec mais $\bar{l}$ ne l'est pas).
- **MaxSatz$_{c*}$** : Dans cette version, la résolution de cycle est appliquée de façon exhaustive à chaque noeud après application de l'UP et des règles d'inférence (règle 1, règle 2, règle 3 et la règle 4), et avant d'appliquer la détection de littéral d'échec. L'application est exhaustive car aucun sous-ensemble de clauses binaires correspondant à une structure de cycle ne reste dans l'instance courante. Notons que le pattern $\bar{l}_1 \vee l_2$, $\bar{l}_1 \vee l_3$, $\bar{l}_2 \vee \bar{l}_3$ est relativement facile à détecter. On fixe un littéral l_1 à vrai et, parmi les clauses unitaires $l_2, l_3, ..., l_k$, on cherche des clauses binaires contenant deux littéraux $\bar{l}_i$ et $\bar{l}_j$ ($2 \leq i < j \leq k$), ce qui peut être efficacement implémenté en utilisant un algorithme de filtrage sur des listes de clauses binaires contenant $\bar{l}_i$ ou $\bar{l}_j$. Il est à noter que la résolution de cycle n'est pas combinée à la détection de littéraux d'échec dans

MaxSatz$_{c*}$.

– **MaxSatz$_c^p$** : Il s'agit d'une version de MaxSatz$_c$ dans laquelle la résolution de cycle est appliquée de façon exhaustive comme un prétraitement, au noeud racine de l'arbre de recherche, et est appliquée comme dans MaxSatz$_c$ dans le reste des noeuds. Remarquons que le "prétraitement" n'a aucun impact sur une instance ne contenant pas de clauses binaires (par exemple, les instances Max-3SAT). Dans ce cas, MaxSatz$_c^p$ n'est que MaxSatz$_c$. Bien que tous les sous-ensembles de clauses binaires correspondant à une structure de cycle sont remplacés dans le prétraitement, la résolution de cycle peut toujours être appliquée au cours de la recherche, car (i) les clauses ternaires peuvent devenir binaires et (ii) les règle 1, règle 2, règle 3 et règle 4 qui s'appliquent dans UP, avant la détection de littéraux d'échec, peuvent transformer des clauses binaires et ajouter des clauses ternaires. Par exemple, si ϕ contient les clauses $x_1 \vee x_2$, $x_1 \vee x_3$, $x_2 \vee x_3$, $\bar{x}_2$ et $\bar{x}_3$, nous n'avons pas de structure de cycle. En appliquant la règle 1, nous avons une clause vide et $x_1 \vee x_2$, $x_1 \vee x_3$, $\bar{x}_2 \vee \bar{x}_3$. Maintenant, la résolution de cycle peut être appliquée.

– **MaxSatz^p** : C'est MaxSatz muni d'une application exhaustive de la résolution de cycle, mais uniquement au noeud racine, comme un prétraitement. MaxSatz^p est simplement MaxSatz pour les instances (ou formules) ne contenant pas de clauses binaires.

6 Résultats expérimentaux et Analyse

Nous avons mené des expérimentations, d'une part afin de comparer les différentes stratégies entre elles, mais aussi pour les comparer à MAX-DPLL et MiniMaxSat de l'autre part. Nous avons donc utilisé les versions de MAX-DPLL (appelée également *toolbar*) et MiniMaxSat qui ont participé à *Max-SAT Evaluation 2007*[3].

Comme benchmarks, nous avons considéré des instances MaxCUT créées à partir de graphes générés aléatoirement, mais aussi des instances générées aléatoirement de Max-2SAT, Max-3SAT et toutes les instances de la catégorie Max-SAT non pondéré de *Max-SAT Evaluation 2007*. Les expérimentations ont été effectuées sur un cluster Linux où les noeuds ont pour processeur : 2GHz AMD Opteron avec 1Go de RAM, sauf pour les instances aléatoires Max-3SAT dont les expérimentations ont été réalisées sur un MacPro 2,8 GHz avec deux Quad-Core Intel Xeon processeurs et 4 Go de RAM

Nous avons comparé tous les solveurs sur les trois types d'instances MaxCUT, Max-2SAT et Max-3SAT. Ainsi, nous avons résolu des ensembles de 100 instances (par nombre de clauses et par type d'instances). Pour les instances Max-2SAT, nous avons considéré 120 variables et

[3]http ://www.maxsat07.udl.es/

FIG. 1 – Temps moyen pour MaxCUT

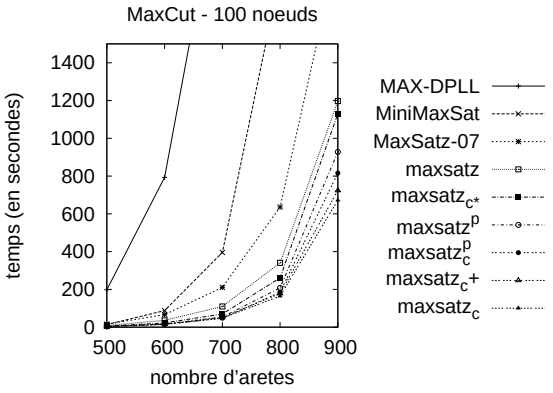


FIG. 2 – Nombre moyen d'applications de la résolution de cycle (excepté les règles 3 et 4) pour MaxCUT

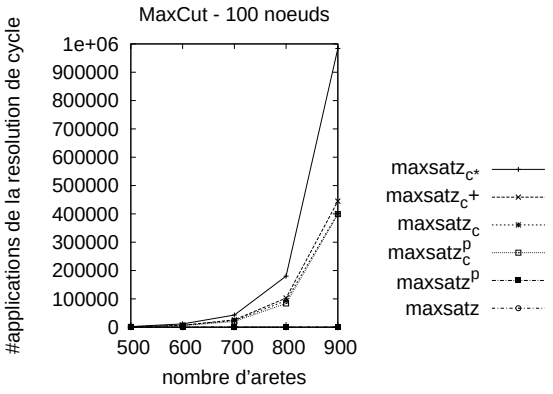


FIG. 3 – Nombre moyen de backtracks pour MaxCUT

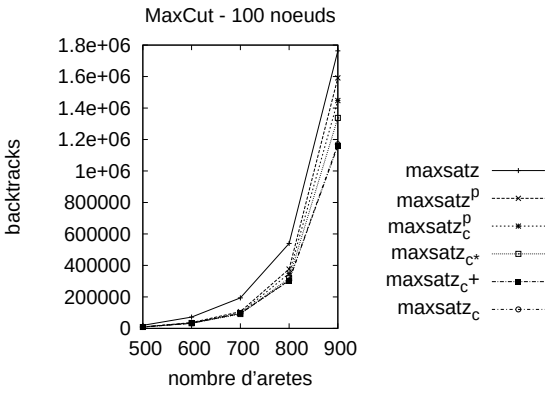


FIG. 4 – Temps moyen pour Max-2SAT

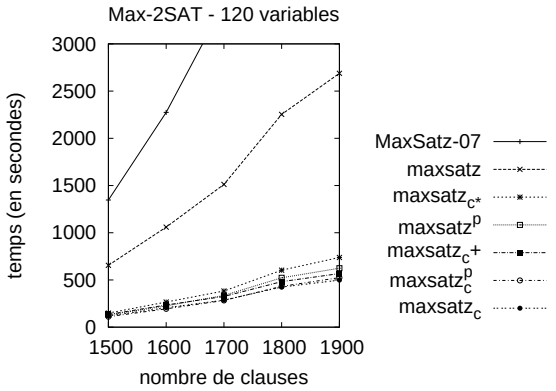


FIG. 5 – Nombre moyen d'applications de la résolution de cycle (excepté les règles 3 et 4) pour Max-2SAT

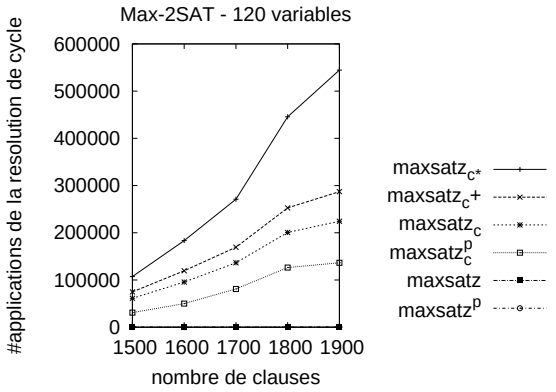


FIG. 6 – Nombre moyen de backtracks pour Max-2SAT

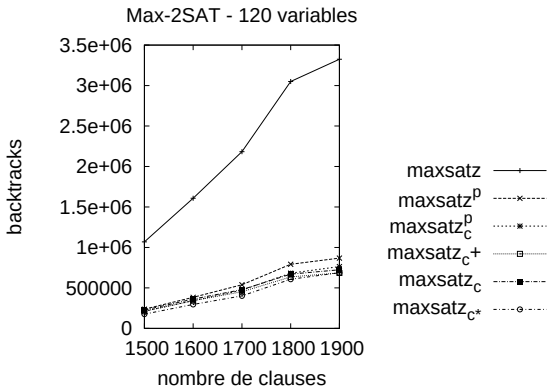


FIG. 7 – Temps moyen pour Max-3SAT

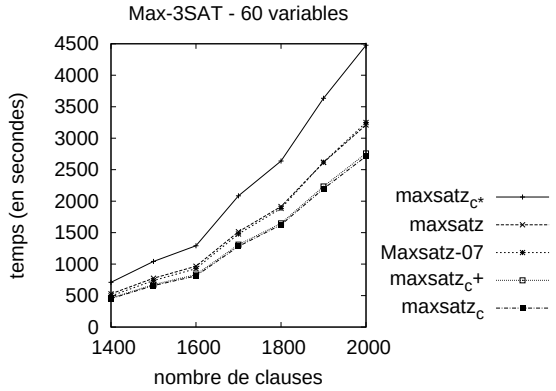


FIG. 8 – Nombre moyen d'applications de la résolution de cycle (excepté les règles 3 et 4) pour Max-3SAT

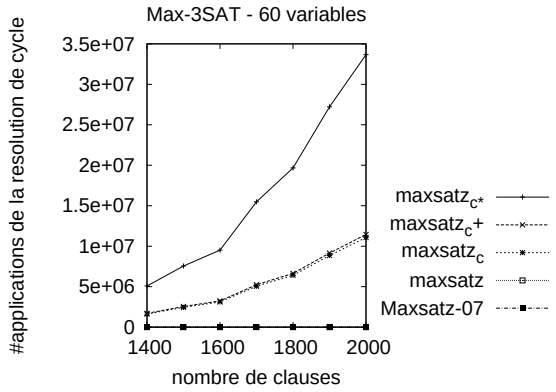
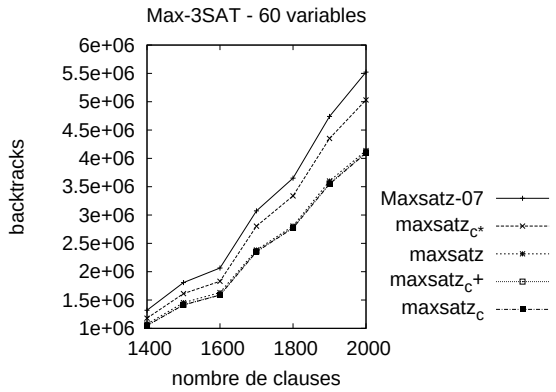


FIG. 9 – Nombre moyen de backtracks pour Max-3SAT



fait varier le nombre de clauses de 1500 à 1900. De manière analogue, les instances Max-3SAT ont eu un nombre de clauses s'échelonnant de 1400 à 2000 pour 60 variables. Quant aux instances MaxCUT, nous leur avons considéré des graphes aléatoires de 100 noeuds, pour un nombre d'arêtes variant de 500 à 900. Ce qui fait des instances Max-SAT de 100 variables avec un nombre de clauses compris entre 1000 et 1800.

Les résultats sont présentés dans les figures numérotées de 1 à 9. La légende est affichée dans le même ordre des performances (courbes) que les solveurs comparés. L'axe horizontal représente le nombre de clauses (pour les instances MaxCUT, le nombre d'arêtes correspond à la moitié du nombre de clauses) et l'axe vertical est la moyenne du paramètre (temps, nombre d'applications de résolution de cycle combinée à la détection des littéraux d'échec ou nombre de backtracks ; selon la figure) nécessaire à un solveur pour résoudre une instance de la série. Les applications des règles 3 et 4 ne sont pas comptées.

Les résultats de MAX-DPLL et MiniMaxSat ne sont pas affichés dans la figure 4 car les deux solveurs sont trop lents, pour Max-2SAT, par rapport aux autres solveurs. Les résultats de MAX-DPLL ne sont pas affichés dans la figure 7 pour la même raison. Notons que pour MAX-DPLL et MiniMaxSat, seule la performance en temps a été montrée.

Pour les instances Max-3SAT, nous n'avons pas montré les résultats de MaxSatz$_c^p$ et MaxSatz^p puisque le prétraitement n'a aucun impact sur Max-3SAT.

Dans les précédentes expérimentations, nous avons observé que MaxSatz$_c$ et MaxSatz$_{c*}$ créent des arbres de tailles similaires (voir les figures backtracks), ce qui signifie que l'application exhaustive de la résolution de cycle dans MaxSatz$_{c*}$ ne contribue pas à l'amélioration de la LB ni à réduire la taille de l'arbre de recherche par rapport à MaxSatz$_c$.

Nos expérimentations montrent clairement que le nombre d'applications de la résolution de cycle peut augmenter vertigineusement sans grand intérêt à cause d'une application exhaustive (comme dans MaxSatz$_{c*}$) comparé à une application plus "modérée" (comme dans MaxSatz$_c$). Par exemple, la figure 8 (Max-3SAT pour 2000 clauses) montre que MaxSatz$_{c*}$ applique un peu moins de 34 millions de fois la résolution de cycle tandis que MaxSatz$_c$ l'applique environs 3 fois moins. Et ceci avec un net écart en temps d'exécution ; allant d'environs 2700 secondes pour MaxSatz$_c$ à 4500 secondes pour MaxSatz$_{c*}$ (voir la figure 7, point 2000 clauses). Situation similaire dans l'exemple de la figure 5 (Max-2SAT), mais avec un écart moindre en performance (voir la figure 4).

Les solveurs de la famille MaxSatz sont plus performants que le reste des solveurs, et à l'intérieur de cette famille, les plus mauvais solveurs sont ceux ayant le moins (MaxSatz) et ceux ayant le plus (MaxSatz$_{c*}$) appliqué la résolution de cycle. En particulier, l'application exhaustive de la résolution de cycle ralentit considérablement MaxSatz sur les instances Max-3SAT. En effet, MaxSatz$_c$ est systématiquement le solveur le plus performant parmi les solveurs comparés.

Nous avons également comparé, sur toutes les instance *Max-SAT Evaluation 2007*, les solveurs suivants : MaxSatz-07, MiniMaxSat, MAX-DPLL, MaxSatz, MaxSatz$_c$ et MaxSatz$_{c*}$. Nous avons utilisé une limitation de temps de 1800 secondes comme dans l'Évaluation. Les résultats obtenus sont présentés au tableau 1, où la première colonne contient la nomenclature de l'instance, la deuxième colonne, le nombre d'instances de chaque série et le reste des colonnes représente la moyenne du temps nécessaire à chaque solveur pour résoudre les instances avec la limitation de temps. Le nombre d'instances résolues, dans la limitation de temps, est affiché entre parenthèses. Le résultat en gras veut dire que le solveur est le plus performant pour l'instance courante, et celui en italique signifie que le solveur est en seconde position en performance.

Nous constatons que les solveurs de la famille MaxSatz dépassent en performance le reste des solveurs. Nous pouvons notamment remarquer que le solveur MaxSatz$_c$ n'est absent qu'une seule fois de la première et deuxième position à la fois[4], et ceci pour toutes les instances confondues, sauf pour les instances DIMACS-MOD où c'est MaxSatz qui est second. Notons que MaxSatz$_c$ améliore globalement MaxSatz en combinant la détection de littéraux d'échec à la résolution de cycle. Aussi, on peut noter que pour les instances Max-3SAT, plus le rapport (nombre de clauses sur nombre de variables) est grand, plus la performance des solveurs de la famille MaxSatz est meilleure. Ce qui explique l'écart en performance constaté entre les instances Max-3SAT du tableau et celles de la figure 7.

Généralement, ces résultats représentent une preuve empirique qu'il est préférable d'appliquer la résolution de cycle de façon limitée mais "raisonnée", comme dans MaxSatz$_c$, que de l'appliquer de façon exhaustive comme dans MaxSatz$_{c*}$. Néanmoins, nous n'avons pas observé d'aussi grandes différences de performance sur les autres expérimentations puisque ces instances sont, en général, relativement faciles pour la famille des solveurs MaxSatz.

7 Conclusions

Les solveurs modernes et rapides BnB pour Max-SAT ont généralement deux composantes pour le calcul de la LB : la composante de sous-estimation et celle de l'inférence. La composante sous-estimation détecte autant que possible des sous-formules disjointes et inconsistantes, et la composante inférence applique des règles d'inférence pour transformer une instance en une autre qui lui est équivalente. Nous avons étudié la combinaison de la *détection*

[4] Uniquement pour les instances RAMSY.

TAB. 1 – **Instances de *Max-SAT Evaluation 2007***

Type d'instance	#I	MaxSatz-07	MiniMaxSat	MAX-DPLL	maxsatz	maxsatz$_c$	maxsatz$_{c*}$
MAX3SAT/40VARS	40	*1.05 (40)*	3.34 (40)	7.20 (40)	1.12(40)	**1.02 (40)**	1.75 (40)
MAX3SAT/50VARS	40	*5.90 (40)*	25.79(40)	57.64 (40)	6.68 (40)	**5.76 (40)**	9.87 (40)
MAX3SAT/60VARS	40	**14.24(40)**	77.53(38)	272.90 (40)	17.20(40)	*16.07(40)*	25.40(40)
MAX3SAT/70VARS	40	**48.82(40)**	207.90(35)	334.33 (29)	63.49(40)	*59.90(40)*	94.38(40)
SPINGLASS	20	69.40(20)	**4.56 (20)**	24.02 (10)	21.44(20)	*21.23(20)*	34.36(20)
RAMSEY	48	*8.99 (34)*	29.81(34)	**20.40 (35)**	13.59(34)	13.57(34)	19.79(34)
MAX2SAT/100VARS	110	1.78 (110)	9.62 (110)	29.02 (110)	1.30 (110)	**0.44 (110)**	*0.59 (110)*
MAX2SAT/140VARS	110	29.57(110)	121.54(99)	235.40 (96)	*19.57(110)*	**5.72 (110)**	4.89 (109)
MAX2SAT/60VARS	110	0.03 (110)	0.19 (110)	0.69 (110)	*0.02 (110)*	**0.01 (110)**	0.02 (108)
MAX3SAT/40VARS	50	*1.50 (50)*	5.53 (46)	9.54 (50)	1.58 (50)	**1.47 (50)**	2.44 (50)
MAX3SAT/60VARS	50	**23.33(50)**	111.81(50)	339.96 (48)	27.28(50)	*25.37(50)*	40.20(50)
MAX3SAT/80VARS	50	**197.58(49)**	230.82(37)	241.94 (28)	227.87(48)	*215.91(48)*	257.16(46)
MAXCUT/DIMACS_MOD	62	**83.86(52)**	100.06(48)	127.82(48)	*92.42(52)*	103.62(52)	122.31(50)
MAXCUT/RANDOM	40	5.58 (40)	15.88(40)	55.54(40)	*4.56 (40)*	**4.54 (40)**	8.34 (40)
MAXCUT/SPINGLASS	5	44.96(3)	**1.62 (3)**	4.75 (2)	13.34(3)	*13.25(3)*	21.24(3)

de littéral d'échec à l'application de la *résolution de cycle*. Nous avons trouvé que l'application exhaustive de résolution de cycle n'est pas aussi efficace que l'on puisse espérer, et que lorsque la propagation unitaire détecte deux littéraux d'échec complémentaires, la résolution de cycle peut être appliquée, réduisant ainsi le nombre de clauses dans la sous-formule inconsistante courante et faisant apparaître deux nouvelles clauses ternaires, celles-ci serviront aux futures applications de la détection de littéral d'échec. Ainsi, la *résolution de cycle* combinée à la *détection de littéral d'échec* utilise une bonne stratégie permettant de détecter beaucoup plus de sous-formules inconsistantes dans la composante sous-estimation du solveur. Ces idées ont été implémentées dans un nouveau solveur appelé MaxSatz$_c$. Les résultats expérimentaux ont clairement montré que MaxSatz$_c$ améliore substantiellement MaxSatz-07 sur de nombreuses et difficiles instances Max-SAT.

Références

[1] Alsinet, T. ;Manyà, F. ; and Planes, J. 2005. Improved exact solver for weighted Max-SAT. In SAT-2005, 371-377.

[2] Bansal, N., and Raman, V. 1999. Upper bounds for MaxSat : Further improved. In ISAAC'99, 247-260.

[3] Bonet,M. L. ; Levy, J. ; and Manyà, F. 2007. Resolution for Max-SAT. Artificial Intelligence 171(8-9) :240-251.

[4] Darras, S. ; Dequen, G. ; Devendeville, L. ; and Li, C. M. 2007. On inconsistent clause-subsets for max-sat solving. In CP-2007, 225-240.

[5] Heras, F., and Larrosa, J. 2006. New inference rules for efficient Max-SAT solving. In AAAI-2006, 68-73.

[6] Heras, F. ; Larrosa, J. ; and Oliveras, A. 2007. MiniMaxSat : A new weighted Max-SAT solver. In SAT-2007.

[7] Larrosa, J., and Heras, F. 2005. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In IJCAI-2005, 193-198.

[8] Larrosa, J. ; Heras, F. ; and de Givry, S. 2008. A logical approach to efficient max-sat solving. Artificial Intelligence 172(2-3) :204-233.

[9] Li, C. M. ; Manyà, F. ; and Planes, J. 2005. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In CP-2005, 403-414.

[10] Li, C. M. ; Manyà, F. ; and Planes, J. 2006. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In AAAI-2006, 86-91.

[11] Li, C. M. ; Manyà, F. ; and Planes, J. 2007. New inference rules for Max-SAT. J. of Artificial Intelligence Research 30 :321-359.

[12] Pipatsrisawat, K., and Darwiche, A. 2007. Clone : Solving weightedmax-sat in a reduced search space. In AI-07, 223-233.

[13] Ramirez, M., and Geffner, H. 2007. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In CP-2007, 605-619.

[14] Wallace, R. J., and Freuder, E. 1996. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximumsatisfiability problems. In Cliques, Coloring and Satisfiability, 587-615.

[15] Xing, Z., and Zhang, W. 2005. An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence 164(2) :47-80.