Amélioration de l'apprentissage des clauses par symétrie dans les solveurs SAT

Belaïd Benhamou, Tarek Nabhani, Richard Ostrowski, Mohamed Réda Saïdi

Amélioration de l'apprentissage des clauses par symétrie dans les solveurs SAT

Belaïd Benhamou **Tarek Nabhani** **Richard Ostrowski** **Mohamed Réda Saïdi**[1]

[1] Université de Provence
Laboratoire des Sciences de l'Information et des Systèmes (LSIS)
Centre de Mathématiques et d'Informatique.
39, rue Joliot Curie - 13453 Marseille cedex 13, France.
{benhamou, nabhani, ostrowski, saidi}@cmi.univ-mrs.fr

Résumé

Le problème de satisfiabilité (SAT) est le premier problème de décision à avoir été montré NP-complet. Il est central en théorie de la complexité. Une formule mise sous forme CNF contient un nombre intéressant de symétries. En d'autres termes, la formule reste invariante si l'on permute quelques variables. De telles permutations sont les symétries de la formule et leurs éliminations peuvent conduire à une preuve plus courte pour la satisfiabilité. D'autre part, de nombreuses améliorations ont été apportées dans les solveurs actuels. Les solveurs de type CDCL sont aujourd'hui capables de résoudre de manière efficace des problèmes industriels de très grande taille (en nombre de variables et de clauses). Ces derniers utilisent des structures de données paresseuses, des politiques de redémarrage et apprennent de nouvelles clauses à chaque échec au cours de la recherche. Bien que l'utilisation des symétries et l'apprentissage de clauses s'avèrent être des principes puissants, la combinaison des deux n'a encore jamais été exploitée. Dans cet article, nous allons montrer comment la symétrie peut être utilisée afin d'améliorer l'apprentissage dans des solveurs de type CDCL. Nous avons mis en application l'apprentissage par symétries dans MiniSat et nous l'avons expérimenté sur différents problèmes. Nous avons comparé MiniSat avec et sans apprentissage par symétries. Les résultats obtenus sont très encourageants et montrent que l'utilisation des symétries dans l'apprentissage est profitable pour des solveurs à base de CDCL.

Abstract

The satisfiability problem (SAT) is shown to be the first decision NP-complete problem. It is central in complexity theory. A CNF formula usually contains an interesting number of symmetries. That is, the formula remains invariant under some variable permutations. Such permutations are the symmetries of the formula, their elimination can lead to make a short proof for a satisfiability proof procedure. On other hand, many improvements had been done in SAT solving, Conflict-Driven Clause Learning (CDCL) SAT solvers are now able to solve great size and industrial SAT instances efficiently. The main theoretical key behind these modern solvers is, they use lazy data structures, a restart policy and perform clause learning at each fail end point in the search tree. Although symmetry and clause learning are shown to be powerful principles for SAT solving, but their combination, as far as we now, is not investigated. In this paper, we will show how symmetry can be used to improve clause learning in CDCL SAT solvers. We implemented the symmetry clause learning approach on the MiniSat solver and experimented it on several SAT instances. We compared both MiniSat with and without symmetry and the results obtained are very promising and show that clause learning by symmetry is profitable for CDCL SAT solvers.

1 Introduction

Krishnamurthy a introduit dans [23] le principe de symétrie dans le calcul propositionnel et a montré que certains problèmes peuvent avoir des preuves plus courtes en augmentant la règle de résolution par les symétries. Les symétries pour les contraintes booléennes ont été étudiées de manière détaillée dans [7, 8, 9].

Les auteurs ont montré comment les détecter et ont prouvé que leur exploitation apporte une amélioration considérable sur plusieurs algorithmes de déduction au-

tomatique. Par la suite, de nombreux travaux sur les symétries ont été développés. Par exemple, l'approche statique utilisée par James Crawford et al. dans [10] pour des théories de logique propositionnelle consiste à ajouter des contraintes exprimant la *symétrie globale* du problème.

Cette technique a été améliorée dans [1] et étendue à la programmation logique en nombres entiers 0-1 dans [2]. Bien que la symétrie ait été introduite en logique propositionnelle, elle a été investie davantage en programmation par contraintes ces dernières années. La notion d'interchangeabilité dans les CSPs est introduite dans [14] et la symétrie pour les CSPs est étudiée plus tôt dans[26, 6].

Puisqu'un grand nombre de contraintes pouvait être ajouté dans l'approche statique, en CSPs, certains chercheurs ont proposé d'ajouter ces contraintes au cours de la recherche. Dans[4, 16, 17], les auteurs enregistrent quelques contraintes conditionnelles qui retirent la symétrie de l'interprétation partielle en cas de retour arrière. Dans[13, 12, 27, 15], les auteurs proposent d'utiliser chaque sous-arbre comme nogood afin d'éviter l'exploration d'interprétations symétriques. Le groupe d'arbres d'équivalence conceptuelle pour l'élimination de valeurs symétriques est introduit dans [28].

Après cela, T. Walsh étudia dans [30] divers propagateurs afin de casser des symétries notamment ceux agissant simultanément sur les variables et les valeurs.

Le problème de satisfiabilité est générique. De nombreux problèmes dans d'autres domaines peuvent se réduire à un test de satisfiabilité par exemple la déduction automatique, la configuration, la planification, l'ordonnancement, etc. . Plusieurs méthodes d'élimination de symétries sont introduites. Parmi elles, les approches statiques[10, 1, 2] qui éliminent les *symétries globales* du problème initial et les approches dynamiques[7, 8, 9] qui détectent et cassent les *symétries locales* à chaque noeud de l'arbre pendant la recherche. Ces deux approches se sont avérées profitables pour la résolution du problème SAT.

D'autre part, de nombreuses améliorations sont apparues ces dernières années autour des solveurs SAT. Ces derniers utilisent de meilleures structures de données, l'apprentissage de clauses [29, 31], le retour arrière non chronologique [29, 21] et différentes politiques de redémarrage [18, 20, 3]. Il a été montré que l'apprentissage est une notion importante [5, 19, 24, 25] qui améliore grandement lâefficacité des solveurs. L'utilisation des symétries est tout aussi utile pour les solveurs SAT mais n'a jamais été utilisée dans l'apprentissage de clauses.

Dans cet article, nous présentons une nouvelle approche pour l'apprentissage qui utilise les symétries du problème. Cette méthode consiste, dans un premier temps, à détecter toutes les symétries globales du problème et de les utiliser lorsqu'une nouvelle clause (clause assertive) est déduite au cours de la recherche. Cela nous permet de déduire toutes les clauses assertives symétriques du problème. L'apprentissage par symétrie est différent des approches consistant à éliminer les symétries globales du problème. Ces dernières sont des méthodes statiques qui ajoutent en prétraitement des clauses afin d'éliminer les interprétations symétriques du problème. L'approche que nous proposons ici est dynamique. Elle génère, au cours de la recherche, toutes les clauses symétriques des différentes clauses assertives afin d'éviter l'exploration de sous-espaces isomorphes. L'avantage de cette méthode est qu'elle n'élimine pas les modèles symétriques comme pour les méthodes statiques. Elle évite d'explorer des sous-espaces correspondant aux no-goods symétriques de l'interprétation partielle courante.

Cet article est organisé de la façon suivante : dans la deuxième partie, nous présentons le contexte des permutations et du problème de satisfiabilité. La troisième partie définit la symétrie et donne les résultats théoriques sur la symétrie que nous utilisons pour l'apprentissage. Nous décrivons ensuite comment utiliser la symétrie dans des solveurs de type CDCL. Nous l'avons ensuite implanté dans un solveur SAT moderne *MiniSat*[11] [1] et nous l'avons évalué sur différents problèmes. La sixième partie synthétise les résultats : nous comparons *MiniSat* avec et sans symétrie lors de l'apprentissage. Enfin, nous apportons une conclusion et présentons quelques-unes des perspectives de ces travaux.

2 Contexte et définitions

2.1 Logique propositionnelle

Nous supposons que le lecteur est familier avec le calcul propositionnel. Nous donnons ici une courte description. Soit V un ensemble de *variables propositionnelles*. Les variables propositionnelles seront distinguées des *littéraux* qui sont des variables propositionnelles avec une valeur d'affection 1 ou 0 qui signifie respectivement $Vrai$ ou $Faux$. Pour une variable propositionnelle ℓ, il y a deux littéraux : le littéral positif ℓ et le négatif $\neg\ell$.

Une $clause$ est une disjonction de littéraux $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$. Une formule $\mathcal{F}$ est mise sous *forme normale conjonctive* (CNF) si et seulement si c'est une conjonction de clauses.

Une *interprétation* d'une CNF $\mathcal{F}$ est une correspondance I définie de l'ensemble des variables de $\mathcal{F}$ dans l'ensemble $\{Vrai, Faux\}$. Nous pouvons considérer I comme l'ensemble des littéraux interprétés. La valeur d'une clause $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ dans *I* est à $Vrai$, si la valeur $Vrai$ est affectée à au moins un de ses littéraux dans I ; sinon, elle est à $Faux$. Par convention, nous définissons la valeur de la *clause vide* ($n = 0$) à $Faux$.

1. L'apprentissage par symétrie est générique. Il peut être utilisé dans n'importe quel solveur qui fait de l'apprentissage

La valeur $I[\mathcal{F}]$ est $Vrai$ si la valeur de chaque clause de $\mathcal{F}$ est à $Vrai$, $Faux$ sinon. Une formule CNF $\mathcal{F}$ est *satisfiable* s'il existe une interprétation I qui affecte la valeur $Vrai$ à $\mathcal{F}$, sinon elle est $insatisfiable$. Dans le premier cas, I est appelé *modèle* de $\mathcal{F}$. Une formule $\mathcal{G}$ est une *conséquence logique* d'une formule $\mathcal{F}$ si $\mathcal{F}$ implique $\mathcal{G}$ et est notée $\mathcal{F} \models \mathcal{G}$. Si une interprétation I est définie seulement sur un sous-ensemble de variables de $\mathcal{F}$, alors elle est appelée *interprétation partielle*. Elle est appelée un *no-good* si $I[\mathcal{F}] = Faux$. Généralement, les solveurs SAT manipulent une interprétation partielle I qui est constituée d'un sous-ensemble de littéraux de décisions D et d'un sous-ensemble de littéraux propagés P par propagation unitaire ($I = D \cup P$). Plus précisement, une interprétation partielle est un produit I=$\prod_{i=1}^{m} \langle (x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i} \rangle$ formé par m littéraux de décisions x_i et tous leurs littéraux propagés ordonnés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$. Le *niveau d'affectation* d'un littéral de décision x_i pour une interprétation I (noté $level(x_i)$) est son ordre d'affectation i dans I, et tous ses littéraux propagés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ ont le même niveau d'affectation i.

2.2 Permutations

Soit $\Omega = \{1, 2, \ldots, N\}$ pour un entier N, où chaque entier peut représenter un littéral d'une formule CNF $\mathcal{F}$. Une permutation de Ω est une correspondante bijective σ de Ω à Ω qui est généralement représentée comme un produit de cycles de permutations. Nous dénotons par $Perm(\Omega)$ l'ensemble de toutes les permutations de Ω et $\circ$ la composition de la permutation de $Perm(\Omega)$. La paire $(Perm(\Omega), \circ)$ forme le groupe de permutation de Ω. En d'autres termes, $\circ$ est fermé et associatif, l'inverse d'une permutation est une permutation et la permutation identique est l'élément neutre. Une paire $(T, \circ)$ forme un sous-groupe de $(S, \circ)$ si et seulement si T est un sous-ensemble de S et constitue un groupe sous l'opération $\circ$.

L'orbite $\omega^{Perm(\Omega)}$ d'un élément ω de Ω sur lequel le groupe $Perm(\Omega)$ agit est $\omega^{Perm(\Omega)}$=$\{\omega^{\sigma} : \omega^{\sigma} = \sigma(\omega), \sigma \in Perm(\Omega)\}$.

Un ensemble générateur du groupe $Perm(\Omega)$ est un sous-ensemble $Gen(\Omega)$ de $Perm(\Omega)$ tel que chaque élément de $Perm(\Omega)$ peut être écrit comme une composition d'éléments de $Gen(\Omega)$. Nous écrivons $Perm(\Omega)$=$< Gen(\Omega) >$. Un élément de $Gen(\Omega)$ est appelé un générateur. L'orbite de $\omega \in \Omega$ peut être calculée à partir de l'ensemble des générateurs $Gen(\Omega)$.

3 Symétrie et apprentissage

3.1 Symétrie

Nous rappelons la définition de la symétrie qui est donnée dans [7, 8]

Définition 1 *Soit $\mathcal{F}$ une formule propositionnelle sous forme CNF et $L_{\mathcal{F}}$ l'ensemble de ses littéraux.* [2]*. Une symétrie de $\mathcal{F}$ est une permutation σ définie sur $L_{\mathcal{F}}$ telle que :*

1. $\forall \ell \in L_{\mathcal{F}}, \sigma(\neg \ell) = \neg \sigma(\ell)$,
2. $\sigma(\mathcal{F}) = \mathcal{F}$

En d'autres termes, une symétrie d'une formule est une permutation des littéraux de cette formule la laissant invariante. Si nous notons par $Perm(L_{\mathcal{F}})$ le groupe de permutations de $L_{\mathcal{F}}$ et par $Sym(L_{\mathcal{F}}) \subset Perm(L_{\mathcal{F}})$ le sous-ensemble de permutations de $L_{\mathcal{F}}$ qui sont les symétries de $\mathcal{F}$, alors $Sym(L_{\mathcal{F}})$ est trivialement un sous-groupe de $Perm(L_{\mathcal{F}})$.

Définition 2 *Soit $\mathcal{F}$ une formule, l'orbite d'un littéral $\ell \in L_{\mathcal{F}}$ sur lequel le groupe de symétries $Sym(L_{\mathcal{F}})$ agit est $\ell^{Sym(L_{\mathcal{F}})}$=$\{\sigma(\ell) : \sigma \in Sym(L_{\mathcal{F}})\}$*

Exemple 1 *Soit $\mathcal{F}$ l'ensemble des clauses suivantes :*
$\mathcal{F}$=$\{a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c\}$
,σ_1 et σ_2 deux permutations définies sur l'ensemble complet $L_{\mathcal{F}}$ de littéraux apparaissant dans $\mathcal{F}$ comme suit :

σ_1=$(a, b, c)(\neg a, \neg b, \neg c)$

σ_2=$(a, \neg a)(b, \neg c)(c, \neg b)$

σ_1 et σ_2 sont deux symétries de $\mathcal{F}$, puisque $\sigma_1(\mathcal{F})$=$\mathcal{F}$=$\sigma_2(\mathcal{F})$. L'orbite du littéral a est $a^{Sym(L_{\mathcal{F}})}$= $\{a, b, c, \neg a, \neg b, \neg c\}$. Nous voyons que tous les littéraux sont dans la même orbite. Par conséquent, ils sont tous symétriques.

La propriété principale d'une symétrie $\sigma \in Sym(L_{\mathcal{F}})$ d'une formule $\mathcal{F}$, est qu'elle conserve l'ensemble de ses modèles. Formellement, B. Benhamou et al [7, 8] ont introduit la propriété suivante :

Proposition 1 *Étant données une formule $\mathcal{F}$ et une symétrie $\sigma \in Sym(L_{\mathcal{F}})$, si I est un modèle de $\mathcal{F}$, alors $\sigma(I)$ est un modèle de $\mathcal{F}$.*

De plus, une symétrie σ transforme chaque no-good I de $\mathcal{F}$ en un no-good $\sigma(I)$.

3.1.1 Détection de symétries

Nous avons utilisé Bliss [22] afin de détecter les symétries d'une formule CNF $\mathcal{F}$. Bliss est un outil pour calculer le groupe d'automorphisme d'un graphe.

Dans [10, 1, 2], les auteurs montrent que chaque formule CNF $\mathcal{F}$ peut être représentée par un graphe $G_{\mathcal{F}}$ construit de la manière suivante :

2. L'ensemble des littéraux L_F contient chaque littéral et sa négation

– chaque variable booléenne est représentée par deux sommets (sommets littéraux) dans $G_{\mathcal{F}}$: le littéral positif et le littéral négatif. Ces deux sommets sont connectés par une arête dans le graphe $G_{\mathcal{F}}$.
– chaque clause dont la longueur est supérieure à deux est représentée par un sommet(un sommet clause). Une arête connecte ce sommet à chaque sommet représentant un littéral de la clause.
– chaque clause binaire est représentée par une arête qui connecte les deux sommets littéraux de cette clause. Nous n'avons pas besoin d'ajouter de sommet clause pour les clauses binaires.

Une propriété importante du graphe $G_{\mathcal{F}}$ est qu'elle préserve le groupe des symétries de $\mathcal{F}$. En d'autres termes, le groupe des symétries $Sym(L_F)$ de la formule $\mathcal{F}$ est identique au groupe d'automorphisme $Aut(G_{\mathcal{F}})$ de sa représentation graphique $G_{\mathcal{F}}$. Pour cela, nous utilisons Bliss sur $G_{\mathcal{F}}$ afin de détecter le groupe de symétries $Sym(L_F)$ de $\mathcal{F}$. Bliss retourne un ensemble de générateurs $Gen(Aut(G_{\mathcal{F}}))$ du groupe d'automorphisme duquel nous pouvons déduire toutes les symétries de $\mathcal{F}$. Bliss offre la possibilité de colorer les sommets du graphe. Un sommet peut être permuté avec un autre sommet de la même couleur. Deux couleurs sont utilisées dans $G_{\mathcal{F}}$, une pour les sommets correspondant aux clauses de $\mathcal{F}$ et une autre pour les littéraux de $L_{\mathcal{F}}$. Cela permet de distinguer les sommets clauses des sommets littéraux et de prévenir ainsi des permutations entre des clauses et des littéraux. Les sources de Bliss sont disponibles à l'adresse (http ://www.tcs.hut.fi/Software/bliss/index.html).

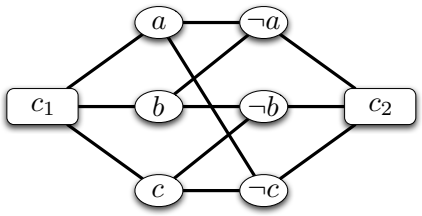

FIGURE 1 – Le graphe $G_{\mathcal{F}}$ correspondant à $\mathcal{F}$

Considérons par exemple la formule $\mathcal{F}$ donnée dans l'exemple 1. Son graphe associé $G_{\mathcal{F}}$ est donné à la figure 1. Les sommets clauses sont représentés par des boîtes et les sommets littéraux par des ellipses. Le groupe d'automorphismes de $G_{\mathcal{F}}$ est identique au groupe des symétries de $\mathcal{F}$. Nous pouvons noter que les deux symétries σ_1 et σ_2 de la formule $\mathcal{F}$ de l'exemple 1 peuvent être vues comme des automorphismes du graphe $G_{\mathcal{F}}$ correspondant.

3.2 Apprentissage par symétries (SLS)

La clé principale des solveurs SAT est la *propagation unitaire*. Étant donné une formule $\mathcal{F}$, nous écrivons $\mathcal{F} \vdash_U \ell$ pour indiquer que le littéral ℓ peut être dérivé de $\mathcal{F}$ en utilisant la *propagation unitaire*. Une formule CNF $\mathcal{F}$ est U-inconsistant si et seulement si $\mathcal{F} \vdash_U Faux$, sinon $\mathcal{F}$ est U-consistent. Maintenant, nous pouvons résumer les définitions d'une clause assertive, d'un littéral assertif et le niveau assertif d'une clause.

Définition 3 *Soit $\mathcal{F}$ une formule CNF, $c = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k \vee x$ une clause de $\mathcal{F}$, et I une interprétation partielle de $\mathcal{F}$ avec m littéraux de décision. La clause c est une clause assertive si et seulement si $I[c] = Faux$, $level(x) = m$ et $\forall i \in \{1, \ldots, k\}$, $level(\ell_i) < m$. Le littéral x est le littéral assertif de c et le niveau assertif de c est le plus haut niveau de ses autres littéraux (le littéral assertif n'est pas inclus).*

Nous introduisons le lemme suivant que nous utiliserons afin de prouver nos résultats sur les clauses assertives symétriques.

Lemme 1 *Soit $\mathcal{F}$ une formule CNF et σ une symétrie de $\mathcal{F}$. Si I=$\langle(x), y_1, y_2, \ldots, y_n\rangle$ est une interprétation partielle constitué par le seul littéral de décision x et tous ses littéraux propagés ordonnés $y_1, y_2, \ldots, y_n$, alors $\sigma(I)$=$\langle(\sigma(x)), \sigma(y_1), \sigma(y_2), \ldots, \sigma(y_n)\rangle$, $level(x)$ dans I est identique à $level(\sigma(x))$ dans $\sigma(I)$, et $\forall i \in \{1, \ldots, n\}$, $level(y_i)$ dans I est identique à $level(\sigma(y_i))$ dans $\sigma(I)$.*

***Preuve* 1** *Pour montrer ceci, nous devons prouver que si $\mathcal{F} \wedge x \vdash_U y_i$,*

alors nous avons $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_i)$ pour tout $i \in \{1, \ldots, n\}$.

Nous le prouvons par induction sur i. Pour $i = 1$ nous devons montrer que si

$\mathcal{F} \wedge x \vdash_U y_1$ alors $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_1)$.

Nous avons $\mathcal{F} \wedge x \vdash_U y_1$ par hypothèse. Cela signifie qu'il existe une clause $c \in \mathcal{F}$

tel que $c = \neg x \vee y_1$. Comme σ est une symétrie de $\mathcal{F}$, alors $\sigma(c) = \neg\sigma(x) \vee \sigma(y_1)$ est une clause de $\mathcal{F}$. Cela implique que $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_1)$.

Supposons maintenant que la propriété est vraie jusque $i - 1$, nous devons montrer qu'elle est vraie pour i. Nous supposons que $\mathcal{F} \wedge x \vdash_U y_i$, et devons montrer que $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_i)$.

De $\mathcal{F} \wedge x \vdash_U y_i$ nous déduisons qu'il existe une clause $c \in \mathcal{F}$ telle que $c = \alpha \vee y_i$ où $\alpha \subseteq \neg x \vee \neg y_1 \vee \neg y_2 \vee \cdots \vee \neg y_{i-1}$. Par conséquent $\sigma(c) = \sigma(\alpha) \vee \sigma(y_i)$ est une clause de $\mathcal{F}$ telle que $\sigma(\alpha) \subseteq \neg\sigma(x) \vee \neg\sigma(y_1) \vee \neg\sigma(y_2) \vee \cdots \vee \neg\sigma(y_{i-1})$. Comme $\mathcal{F} \wedge x \vdash_U y_i$, alors $\mathcal{F} \wedge x \vdash_U y_j, \forall j \in \{1, \ldots, i-1\}$. Par hypothèse, nous avons $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_j), \forall j \in \{1, \ldots, i-1\}$. Il est maintenant trivial que $\mathcal{F} \wedge \sigma(x) \vdash_U \sigma(y_i)$. Il est aussi évident que $level(x)$ dans I est identique à $level(\sigma(x))$ dans $\sigma(I)$, et $\forall i \in \{1, \ldots, n\}$, $level(y_i)$ dans I est identique à $level(\sigma(y_i))$ dans $\sigma(I)$.

Du lemme précédent, nous pouvons déduire la proposition suivante.

Proposition 2 *Étant données une formule CNF $\mathcal{F}$ et une symétrie σ de $\mathcal{F}$, si I=$\prod_{i=1}^{m}\langle(x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}\rangle$ est une interprétation partielle composée de m littéraux de décisions x_i et tous les littéraux ordonnés associés propagés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$ par propagation unitaire, nous avons alors $\sigma(I)$=$\prod_{i=1}^{m}\langle(\sigma(x_i)), \sigma(y_{i,1}), \sigma(y_{i,2}), \ldots, \sigma(y_{i,k_i})\rangle$, et $\forall x \in I, level(x)$ in I est identique à $level(\sigma(x))$ dans $\sigma(I)$.*

***Preuve* 2** *La preuve peut être dérivée par application du précédent lemme de manière récursive sur les littéraux de décision x_i et leurs littéraux propagés associés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$.*

Maintenant, nous introduisons la principale propriété sur la symétrie que nous utiliserons pour améliorer l'apprentissage dans les solveurs SAT de type CDCL.

Proposition 3 *Étant données une CNF $\mathcal{F}$, une symétrie σ de $\mathcal{F}$, et une interprétation partielle I=$\prod_{i=1}^{m}\langle(x_i), y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}\rangle$ constituée de m littéraux de décision x_i et leurs littéraux ordonnés propagés $y_{i,1}, y_{i,2}, \ldots, y_{i,k_i}$. Si $c = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k \vee x$ est une clause assertive correspondant à I avec x son littéral assertif, alors $\sigma(c)$ est une clause assertive correspondant à l'interprétation partielle symétrique $\sigma(I)$ avec $\sigma(x)$ comme littéral assertif.*

***Preuve* 3** *Nous devons montrer que $\sigma(I)[\sigma(c)] = Faux$, $level(\sigma(x)) = m$ et $\forall i \in \{1, \ldots, k\}$, $level(\sigma(\ell_i)) < m$ dans $\sigma(I)$. Par hypothèse, c est une clause assertive correspondant à I, donc $I[c] = Faux$. Comme σ est une symétrie, on a $\sigma(I)[\sigma(c)] = Faux$. De la proposition 2, nous pouvons vérifier la condition du niveau : $level(\sigma(x)) = m$ et $\forall i \in \{1, \ldots, k\}$ $level(\sigma(\ell_i)) < m$ dans $\sigma(I)$.*

Cette propriété permet aux solvers CDCL d'ajouter en même temps la clause assertive par rapport à I et toutes les clauses assertives symétriques $\sigma(c)$. Cela permet d'éviter de parcourir des sous-espaces isomorphiques correspondant à l'interprétation partielle symétrique $\sigma(I)$ de I. Les expérimentations qui vont suivre montreront que cette propriété améliore considérablement l'efficacité des solveurs CDCL.

Le cas le plus intéressant est lorsque la clause assertive est réduite à un mono-littéral. Nous pouvons déduire par symétrie tous les littéraux appartenant à l'orbite de ce mono-littéral et ainsi propager directement tous les littéraux opposés de cet orbite. Formellement, nous avons la propriété suivante :

Proposition 4 *Étant données une CNF $\mathcal{F}$ et une interprétation partielle I, si ℓ est une clause assertive unitaire, alors pour tout $\ell' \in \ell^{Sym(L_{\mathcal{F}})}$, ℓ' est une clause assertive unitaire et $\mathcal{F}$ est satisfiable si et seulement si $(\mathcal{F} \wedge \neg\ell \bigwedge_{\ell_i \in \ell^{Sym(L_{\mathcal{F}})}} \neg\ell_i)$ est satisfiable.*

***Preuve* 4** *La proposition est un cas particulier de la proposition 2.*

Le schéma d'apprentissage par symétrie (SLS) est alors un schéma classique d'apprentissage (SL) incluant les deux propriétés sur les symétries qui permettent d'inférer des clauses assertives symétriques afin de booster l'apprentissage. Dans ce qui suit, nous montrons comment ce schéma est implanté dans un solveur CDCL.

4 Avantages de la symétrie dans des algorithmes de recherche

Dans cette partie, nous présentons la façon dont le nouveau schéma d'apprentissage par symétrie (SLS) peut être ajouté aux solveurs CDCL. Ce nouveau solveur moderne est basé sur la propagation unitaire, l'apprentissage de clauses [29, 31] par symétrie, une politique de redémarrage [18] et du retour arrière non-chronologique [29, 21]. Dans l'algorithme 1 présenté ci-dessous, nous décrivons cette procédure avec l'ajout de la symétrie. Le pseudo-code de cette procédure (appelé SCLR) est basé sur celui présenté dans [25].

Tout le background théorique de base, par exemple, la séquence des points de décicion D, le niveau d'assertion, le retour arrière non-chronologique et les redémarrages sont les mêmes que les algorithmes classiques à base de CDCL [25]. Si $I = D \cup P$ est une interprétation partielle de la formule $\mathcal{F}$, alors l'état correspondant à I dans l'arbre de recherche du solveur SAT considéré est donné par $S = (\mathcal{F}, \Gamma, D)$ où $D = (\ell_1, \ell_2, \ldots, \ell_k)$ est l'ensemble ordonné des littéraux de décision de I et ℓ_i est le littéral de décision au niveau i. Γ est une CNF telle que $\mathcal{F} \models \Gamma$.

Un état $S = (\mathcal{F}, \Gamma, D)$ est U-inconsistant, respectivement U-consistant, si et seulement si $\mathcal{F} \wedge \Gamma \wedge D$ est U-inconsistant, respectivement U-consistant. Les clauses de Γ sont des clauses assertives qui sont ajoutées à la formule initiale $\mathcal{F}$ exprimant les interprétations partielles générées qui sont des no-goods.

La principale différence entre la procédure SCLR que nous proposons et les procédures classiques de type CDCL et l'implantation des deux propositions 3 et 4 dans SCLR (lignes 10 à 14). En effet, lorsqu'il y a une interprétation partielle conflictuelle I et que l'ensemble des points de décision D est non vide, une clause assertive c est déduite et deux cas sont testés : si la clause assertive c est unitaire(ligne 10), alors tous les littéraux symétriques (littéraux de son orbite, ligne 11) sont alors ajoutés à Γ (ligne 14) et leur négation est propagée à la racine de l'arbre de recherche. Si la clause assertive c n'est pas unitaire

(ligne 12), alors c et toutes les clauses symétriques $\sigma(c)$ induites par les générateurs $\sigma \in Gen(Sym(L_F))$ (ligne 13) sont ajoutées à Γ (ligne 14). Pour des raisons d'efficacité, dans notre implantation, nous limitons la génération de clauses symétriques (non unitaires) à celle produites par l'ensemble des générateurs du groupe de symétries de la formule.

Algorithm 1: SCLR : solveurs SAT avec apprentissage par symétries et redémarrages

entrée: Une formule CNF $\mathcal{F}$
sortie : Une solution de $\mathcal{F}$ ou unsat si $\mathcal{F}$ n'est pas satisfiable

1 $D \longleftarrow \langle\rangle$ // Littéraux de décision;
2 $\Gamma \longleftarrow vrai$ // Clauses apprises;
3 **while** $vrai$ **do**
4 **if** $S = (\mathcal{F}, \Gamma, D)$ *est* $U - inconsistant$ **then**
5 // Il y a un conflit.
6 **if** $D = \langle\rangle$ **then**
7 **return** *unsat*
8 $c \longleftarrow$ une clause assertive de S
9 $m \longleftarrow$ le niveau assertif de c
10 **if** c *est unitaire* **then**
11 $A \longleftarrow \{\ell \mid \ell \in c^{Sym(L_{\mathcal{F}})}\}$
12 **else**
13 $A \longleftarrow \{\sigma(c) | \sigma \in Gen(Sym(L_{\mathcal{F}}))\}$
14 $\Gamma \longleftarrow \Gamma \bigwedge_{c \in A} c$
15 $D \longleftarrow D_m$ // les m premières décisions
16 **else**
17 // Pas de conflit.
18 **if** *redémmarage* **then**
19 $D \longleftarrow \langle\rangle$
20 $S = (\mathcal{F}, \Gamma, D)$
21 Choisir un littéral ℓ tel que $S \nvdash \ell$ et $S \nvdash \neg\ell$
22 **if** $\ell = null$ **then**
23 **return** D // satisfiable
24 $D \longleftarrow D, \ell$

Nous avons choisi pour notre implantation le solveur *MiniSat*[11] auquel nous avons ajouté l'apprentissage par symétrie. Cependant, notre approche est générique et peut donc être utilisée dans d'autres solveurs de type CDCL. Nous présentons les résultats expérimentaux dans la partie suivante.

5 Expérimentation

Nous étudions les performances de notre approche par une analyse expérimentale. Nous avons pour cela choisi certaines instances afin de tester notre méthode d'apprentissage par symétrie. Nous nous attendons à ce que l'apprentissage par symétrie soit profitable aux solveurs pour des problèmes réels. Nous avons testé et comparé *MiniSat* et *MiniSat* avec apprentissage par symétrie (*MiniSat+SymCDCL*) sur plusieurs instances.

		Minisat		Minisat+SymCDCL	
Instance	$\#V : \#C$	$Noeuds$	$Temps$	$Noeuds$	$Temps$
chnl10_12	240 : 1344	2009561	67.26	28407	1.58
chnl10_13	260 : 1586	3140061	128.68	29788	1.92
chnl11_12	264 : 1476	− − −	>1200	246518	18.44
chnl11_13	286 : 1742	− − −	>1200	185417	15.20
chnl11_20	440 : 4220	− − −	>1200	61287	7.84
fpga10_8_sat	120 : 448	264	0.00	463	0.00
fpga10_9_sat	135 : 549	250	0.00	494	0.01
fpga12_11_sat	198 : 968	421	0.00	982	0.02
fpga12_12_sat	216 : 1128	403	0.00	343	0.00
fpga12_8_sat	144 : 560	390	0.00	568	0.01
fpga12_9_sat	162 : 684	383	0.00	1089	0.03
fpga13_10_sat	195 : 905	499	0.00	2311	0.06
fpga13_12_sat	234 : 1242	335	0.00	336	0.00
fpga13_9_sat	176 : 759	408	0.00	603	0.01
hole7	56 : 204	10123	0.08	233	0.00
hole8	72 : 297	40554	0.37	8323	0.15
hole9	90 : 415	202160	2.69	15184	0.35
hole10	110 : 561	1437244	27.54	73844	2.10
hole11	132 : 738	23096626	778.46	249897	9.49
hole12	156 : 949	− − −	>1200	837072	43.23
Urq3_5	46 : 470	9403639	79.09	1830	0.04
Urq4_5	74 : 694	− − −	>1200	18442	0.61
Urq5_5	121 : 1210	− − −	>1200	1798674	167

TABLE 1 – Résultats sur quelques instances SAT

D'abord, nous avons exécuté les deux méthodes sur différentes instances comme les $FPGA$ (Field Programmable Gate Array), $Chnl$ (Allocation de fréquences), Urq (Urquhart) and $Hole$ (Pigeon) qui possèdent des symétries. Ensuite, nous testons ces deux méthodes sur des problèmes de coloration de graphes difficiles. Pour finir, différents problèmes des dernières compétitions SAT sont testés. Nous comparons les performances en nombre de noeuds et en temps CPU. Le temps nécessaire pour le calcul des symétries du problème initial est ajouté au temps CPU total pour *MiniSat+SymCDCL*. Le code est écrit en C++ et compilé sur un Pentium 4, 2.8 GHZ avec 1 Gb de RAM.

5.1 Résultats sur différents problèmes symétriques

Le tableau 1 montre la comparaison de notre approche avec *MiniSat* sur des problèmes SAT connus. Les colonnes nous donnent le nom de l'instance, la taille ($\#V/\#C$), le nombre de noeuds de l'arbre de recherche et le temps CPU utilisé pour trouver la solution.

Le tableau 1 montre que notre approche *MiniSat+SymCDCL* est en général meilleure que *MiniSat* tant en temps qu'en nombre de noeuds sur les problèmes $Hole$ et $Chnl$. Pour les instances $FPGA$, nous obtenons des résultats similaires en temps. Ceci est dû au fait que ces pro-

blèmes sont satisfiables et *MiniSat* arrive à trouver une solution rapidement. Les problèmes Urq sont plus difficiles que les problèmes $FPGA$ ou les problèmes $Chnl$. Nous voyons que *MiniSat* n'est capable de résoudre que le premier problème alors que notre approche permet de les résoudre tous efficacement.

5.2 Résultats sur les problèmes de coloration de graphe

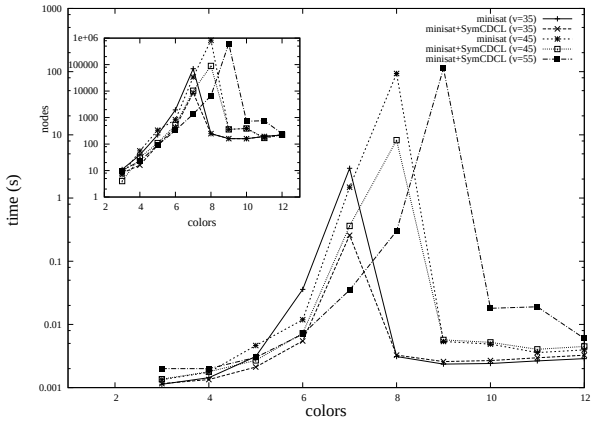

FIGURE 2 – Courbes des noeuds et temps CPU de *MiniSat* et *MiniSat+SymCDCL* sur des problèmes de coloration de graphe générés aléatoirement pour $n = \{35, 45, 55\}$ et $d = 0.5$

Les problèmes de coloration de graphe aléatoires sont générés selon les paramètres suivants :

(1) le nombre de sommets du graphe (n), (2) le nombre de couleurs ($Colors$) et (3) la densité du graphe (d), qui est un réel compris entre 0 et 1 correspondant au ratio du nombre de contraintes du graphe sur le nombre total de contraintes possibles. Les paramètres n, $Colors$ et d étant fixés, nous générons 100 problèmes aléatoirement et les résultats (temps CPU moyen , nombre de noeuds moyen) sont reportés. Le temps limite est fixé à 800 secondes pour résoudre l'instance.

Nous reportons à la figure 2 les résultats pratiques de *MiniSat*, et *MiniSat+SymCDCL*, sur trois classes de problèmes où le nombre de variables (les sommets du graphe) est fixé à 35, 45 et 55 respectivement et où la densité est fixée à ($d = 0.5$). La grande courbe de la figure 2 représente le temps CPU moyen en fonction du nombre de couleurs pour les deux méthodes et pour chaque classe de problème. La petite courbe reporte le nombre moyen de noeuds.

Des deux courbes, nous remarquons qu'en moyenne *MiniSat+SymCDCL* explore moins de noeuds que *MiniSat* et les résout plus rapidement. L'apprentissage de clauses par symétrie est profitable autour de la région difficile de ces problèmes et notre méthode *MiniSat+SymCDCL* peut résoudre des problèmes contenant 55 sommets dans le temps imparti, alors que *MiniSat* n'y arrive pas. Ceci explique que nous ne reportons pas les courbes de *MiniSat* (pour 55 sommets)à la figure 2.

5.3 Résultats sur des problèmes des dernières compétitions SAT

Nous présentons les résultats des deux méthodes (*MiniSat* et *MiniSat+SymCDCL*) sur 180 classes de problèmes issues des dernières compétitions SAT. Le temps CPU limite est fixé à 1200 secondes pour la résolution de chaque problème. Notre premier but est ici de trouver des problèmes contenant des symétries et de regarder le comportement de l'apprentissage par symétrie sur ces derniers. La sélection des problèmes s'est faite en recherchant si le problème comporte des symétries globales. Si c'est le cas, nous la sélectionnons. Sinon nous utilisons le prétraitement de *MiniSat* et nous testons la présence de symétrie sur le problème simplifié. La figure 3 montre la comparaison de *MiniSat+SymCDCL* et *MiniSat* en nombre de noeuds (figure du bas) et en temps CPU (figure du haut) sur les 180 problèmes selectionnés. Sur les deux figures,

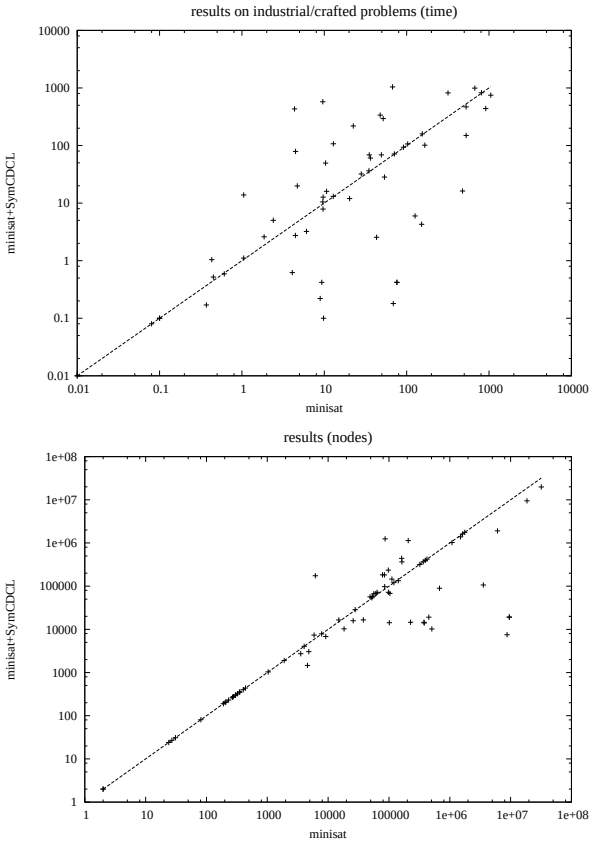

FIGURE 3 – Temps CPU et nombre de noeuds sur quelques problèmes SAT

		Minisat		Minisat+SymCDCL	
Instance	$\#V : \#C$	*Noeuds*	*Temps*	*Noeuds*	*Temps*
cmu-bmc-barrel6	2306 : 8931	101246	4.09	14256	0.62
counting-easier-php-012-010	120 : 672	3565803	151.89	106726	4.28
gus-md5-04	68679 : 223994	3542	6.08	2724	3.22
gus-md5-05	68827 : 224473	4811	20.29	3044	12.00
gus-md5-06	68953 : 224868	18311	53.74	10256	28.35
gus-md5-07	69097 : 225325	37867	165.98	16583	101.22
gus-md5-09	69487 : 226581	103672	1050.09	68221	747.90
gus-md5-10	69503 : 226618	− − −	>1200	97146	1143.27
mod2-3cage-9-12	87 : 232	9533790	76.14	19204	0.42
mod2-3cage-9-14	87 : 232	8754084	68.79	7490	0.18
pmg-11-UNSAT	169 : 562	18562286	911.78	9474789	439.57
Q32inK09	36 : 7938	452083	43.08	19195	2.55
Q32inK10	45 : 38430	380172	126.41	14059	5.98
Q32inK11	55 : 139590	376321	477.15	14543	16.20

TABLE 2 – Résultats sur certains problèmes

l'axe des y (resp. l'axe des x) représente les résultats de *MiniSat+SymCDCL* (resp. *MiniSat*). Un point sur la figure du haut (respectivement figure du bas) représente le temps CPU (respectivement le nombre de noeuds) des deux solveurs pour un problème donné. La projection du point sur l'axe des y (resp. l'axe des x) représente la performance de *MiniSat+SymCDCL* (resp. *MiniSat*) en temps CPU pour la figure du haut et en nombre de noeuds pour la figure du bas. Les points en dessous de la diagonale indiquent que *MiniSat+SymCDCL* est meilleur que *MiniSat*. Les points aux alentours de la diagonale montrent des résultats similaires et les points au dessus indiquent que *MiniSat* est meilleur. Nous voyons à la figure 3 qu'il existe de nombreux problèmes où *MiniSat+SymCDCL* est meilleur que *MiniSat*. Sur certains problèmes, *MiniSat* est meilleur que *MiniSat+SymCDCL* en temps CPU. Cela arrive généralement lorsque le problème admet une solution. *MiniSAT* trouve rapidement celle-ci.

Le tableau 2 donne le détail des résultats des deux méthodes sur certains problèmes issus des dernières compétitions SAT pour lesquels *MiniSat+SymCDCL* est meilleur que *MiniSat*, tant en temps CPU qu'en nombre de noeuds. Nous remarquons que *MiniSat+SymCDCL* résout l'instance *gus-md5-10* alors que *MiniSat* n'y parvient pas dans le temps imparti.

6 Conclusion et perspectives

Dans cet article, nous augmentons l'apprentissage des clauses par symétries. Nous introduisons un nouveau schéma d'apprentissage (SLS) qui peut être utilisé par n'importe quel

solveur SAT à base de CDCL. L'apprentissage par symétrie permet, en plus de l'ajout de la clause assertive dans un noeud de l'arbre de recherche, d'ajouter toutes les clauses assertives symétriques. Considérer les clauses assertives symétriques permet d'éviter aux solveurs SAT d'explorer des sous-espaces isomorphiques. Nous avons implanté le schéma d'apprentissage par symétrie (SLS) dans *MiniSat* et avons expérimenté *MiniSat* et *MiniSat* avec le schéma SLS sur une grande variété de problèmes. Les premiers résultats expérimentaux sont très encourageants, et montrent que l'utilisation de la symétrie dans l'apprentissage est profitable sur la plupart des problèmes testés.

Comme amélioration future, nous chercherons à trouver une bonne stratégie de suppression de clauses qui préservera les clauses symétriques apprises pendant la recherche. Actuellement, *MiniSat* peut supprimer des clauses assertives symétriques qui peuvent être utiles pour couper des sous-espaces isomorphes.

Un autre point est que seules les symétries du problème initial (symétries globales) sont utilisées dans ce schéma d'apprentissage. Nous prévoyons d'étendre ce schéma d'apprentissage à l'exploitation des symétries locales, qui peuvent être détectées au cours de la recherche.

Références

[1] F. A. Aloul, A.Ramani, I. L. Markov, and K. A. Sakallak. Solving difficult sat instances in the presence of symmetry. *In IEEE Transaction on CAD, vol. 22(9)*, pages 1117–1137, 2003.

[2] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallak. Symmetry breaking for pseudo-boolean satisfiabilty. *In ASPDAC'04*, pages 884–887, 2004.

[3] Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In *SAT*, pages 21–27, 2008.

[4] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *Principle and Practice of Constraint Programming - CP'99*, 1999.

[5] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22 :319–351, 2004.

[6] B. Benhamou. Study of symmetry in constraint satisfaction problems. *In Proceedings of the 2nd International workshop on Principles and Practice of Constraint Programming - PPCP'94*, 1994.

[7] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *Eleventh International Conference on Automated Deduction, Saratoga Springs,NY, USA*, 1992.

[8] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *Journal of Automated Reasoning (JAR),*, 12 :89–102, 1994.

[9] B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language. *in proceedings of STACS'94, Caen France*, pages 71–82, 1994.

[10] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates

for search problems. In *KR'96 : Principles of Knowledge Representation and Reasoning*, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[11] Niklas Eén and Niklas S´orensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[12] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 93–108. Springer Verlag, 2001.

[13] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *International conference on constraint programming*, volume 2239 of *LNCS*, pages 77–82. Springer Verlag, 2001.

[14] E.C. Freuder. Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, pages 227–233, 1991.

[15] I. P. Gent, W. Hervey, T. Kesley, and S. Linton. Generic sbdd using computational group theory. In *Proceedings CP'2003*, 2003.

[16] I. P. Gent and B. M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings ECAI'2000*, 2000.

[17] I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints : Symmetry breaking during search. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 415–430. Springer Verlag, 2002.

[18] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *CP*, pages 121–135, 1997.

[19] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively p-simulate general propositional resolution. In *AAAI*, pages 283–290, 2008.

[20] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In *IJCAI'07 : Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2318–2323, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[21] Roberto J. Bayardo Jr. and Robert Schrag. Using csp look-back techniques to solve real-world sat instances. In *AAAI/IAAI*, pages 203–208, 1997.

[22] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In David Applegate, Gerth Stølting Brodal, Daniel Panario, and Robert Sedgewick, editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

[23] B. Krishnamurty. Short proofs for tricky formulas. *Acta informatica*, (22) :253–275, 1985.

[24] Knot Pipatsrisawat and Adnan Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In *AAAI*, pages 1481–1484, 2008.

[25] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers with restarts. In *CP*, pages 654–668, 2009.

[26] J. F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *In J. Kamorowski and Z. W. Ras,editors, Proceedings of ISMIS'93, LNAI 689*, 1993.

[27] J.F. Puget. Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, pages 446–461. Springer Verlag, 2002.

[28] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. A. Linton. Tractable symmetry breaking using restricted search trees. *In, proceedings of ECAI'04*, pages 211–215, 2004.

[29] J. P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

[30] T. Walsh. General symmetry breaking constraints. *In, proceedings of CP'06*, pages 650–664, 2006.

[31] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.