Détection et élimination dynamique de la symétrie dans le problème de satisfiabilité

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

Détection et élimination dynamique de la symétrie dans le problème de satisfiabilité

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 SAT est connu pour être le premier problème de décision de classe NP-complet (Cook,71). C'est un problème central dans la théorie de la complexité. Dans la dernière décennie, les procédures prouvant la satisfiabilité ont été améliorées par l'élimination de la symétrie. Une formule CNF contient usuellement un nombre intéressant de symétries. Il y a deux types d'exploitions des symétries. La première correspond à l'élimination des symétries globales, c'est à dire, seules les symétries initiales du problème (le problème à la racine de l'arbre de recherche) sont détectées et éliminées. La seconde exploite toutes les symétries locales qui apparaissent à chaque noeud de l'arbre de recherche. Les symétries locales doivent être détectées et éliminées dynamiquement durant la recherche. Exploiter ce genre de symétrie semble être une tâche difficile. Quasiment tous les travaux sur l'exploitation de la symétrie dans le problème de la satisfiabilité traitent uniquement le cas des symétries globales. En dépit de leur importance en pratique, seuls quelques travaux étudient les symétries locales.

Détecter et éliminer efficacement les symétries locales durant la recherche est un challenge important. Le travail que nous présentons ici est une contribution pour répondre à ce difficile challenge. Nous présentons une nouvelle méthode pour l'élimination des symétries locales qui consiste à réduire l'instance partielle SAT, non encore résolue correspondante à chaque noeud de l'arbre de recherche, à un graphe dont le groupe d'automorphismes est équivalent au groupe de symétries de l'instance partielle SAT.

Nous avons utilisé l'outil Saucy pour le calcul du groupe d'automorphisme et nous avons implémenté une technique de coupure de symétrie dans un solveur SAT. Nous avons expérimenté cette méthode sur plusieurs instances SAT et nous l'avons comparé à une méthode qui exploite les symétries globales. Les résultats obtenus sont prometteurs. L'exploitation des symétries locales améliore l'exploitation des seules symétries globales dans la résolution de nombreux problèmes difficiles et elles leurs sont complémentaires si nous combinons les deux techniques.

Abstract

The SAT problem is shown to be the first decision NP-complete problem (Cook,71). It is central in complexity theory. In the last decade, the satisfiability proof procedures are improved by symmetry elimination. A CNF formula usually contains an interesting number of symmetries. There are two kinds of symmetry exploitation. The first one corresponds to global symmetry breaking, that is, only the symmetries of the initial problem (the problem at the root of the search tree) are detected and eliminated. The second one deals with all local symmetries that appear at each node of the search tree. Local symmetry has to be detected and eliminated dynamically during the search. Exploiting such symmetries seems to be a hard task. Almost all of the known works on symmetry in satisfiability are on global symmetry. Only few works are carried on local symmetry, despite their importance in practice. An important challenge is then to detect and break local symmetries efficiently during the search. The work that we present here is a contribution towards an answer to this hard challenge. We present a new method for local symmetries breaking that consists in detecting dynamically local symmetries by reducing the remaining partial SAT instance at each node of the search tree to a graph that has an equivalent automorphism group than the symmetry group of the partial SAT instance. We used the software Saucy to compute the automorphism group and implemented a local symmetry cut in a SAT solver. We experimented

this method on several SAT instances and compared it with a method exploiting global symmetries. The results obtained are very promising. Local symmetry improves global symmetry on some hard instances and is complementary to global symmetry.

1 Introduction

Krishnamurthy dans [20] introduit le principe de la symétrie dans le calcul propositionnel et a montré que certaines formules difficiles à prouver peuvent avoir des preuves courtes si on ajoute au système de résolution la règle de symétrie. Les symétries ont été même utilisées bien avant, pour la résolution de nombreux problèmes comme par exemple le problème des huit reines [16]. Elles ont aussi été introduites dans la résolution des problèmes de satisfaction de contraintes [14, 25, 3], dans un intelligent algorithme de Backtracking [9] et dans la logique du premier ordre [11].

La symétrie est devenue une notion importante dans la programmation par contrainte. Durant la dernière décennie, plusieurs travaux sur l'exploitation des symétries dans la résolution des problèmes SAT et CSP sont apparues. Cependant, peu de travaux exploitent la détection et l'élimination dynamique des symétries [4, 5, 6, 7, 15]. La plupart des méthodes n'exploitent que les symétries globales [12, 1, 2], c'est à dire, les symétries du problème initial correspondant à la racine de l'arbre de recherche.

Une symétrie d'une formule logique est une permutation de littéraux qui laisse invariant la formule. Il existe de nombreux problèmes en intelligence artificielle qui contiennent un nombre important de symétries qui s'expriment sous forme de formules CNF.

L'importance de l'exploitation des symétries lors de la résolution peut être mise en évidence sur de nombreux problèmes que les méthodes de résolution classiques ont du mal à résoudre efficacement. Si on prend par exemple le problème des pigeons [8, 17], ou le problème de Ramsey [18]. Les deux problèmes sont connus pour être difficiles à résoudre pour les méthodes de résolution classiques et qu'ils sont représentés en logique du premier ordre par un petit ensemble de formules qui devient très large lorsque l'on tente de calculer toutes les instancions propositionnelles terminales. L'ensemble des clauses propositionnelles obtenues, contient un nombre important de symétries. C'est à dire que, l'ensemble des clauses reste invariant par rapport à plusieurs permutations de variables. Il en résulte que prouver la satisfiabilité par l'exploitation de ces permutations a une complexité polynômiale alors que l'on sait que les méthodes de résolution classiques sont tous de complexité exponentielle pour résoudre ces deux problèmes, si l'élimination de la symétrie n'est pas prise en compte.

Le problème de satisfiabilité est un problème générique, de nombreux problèmes provenant d'autres domaines peuvent être réduit au problème de satisfiabilité. Par exemple, la déduction automatique, la configuration, la planification, l'ordonnancement, etc. Plusieurs méthodes d'élimination de la symétrie pour le problème de satisfiabilité ont été introduites [12, 1, 2]. Cependant, pratiquement toutes ces méthodes exploitent uniquement les symétries globales et ignorent le traitement des symétries locales. Ceci est du à la difficulté de la détection et de l'élimination dynamique de ces symétries. Contrairement aux symétries globales qui peuvent être exploitées par des approches statiques faciles à implémenter.

Une approche qui détecte et élimine dynamiquement les symétries locales en logique propositionnelle est proposée dans [4, 5, 6]. Mais cette méthode est incomplète dans le sens où elle détecte qu'une partie des symétries locales, et non pas tout le groupe total de symétries locales. Une alternative à cette méthode est d'adapter et d'utiliser l'outil Saucy qui permet de calculer les automorphismes de graphes [1] afin de détecter les symétries locales durant la recherche, puisque le groupe d'automorphismes du graphe déduit à partir de l'instance SAT est identique au groupe de symétries de l'instance SAT.

Dans ce papier, nous présentons une méthode alternative qui élimine les symétries locales pour la résolution du problème SAT, qui exploite le groupe total de symétries. Cette méthode consiste à réduire incrémentalement la sous-formule logique définie à chaque noeud de l'arbre de recherche à un graphe sur lequel nous utilisons un outil de calcul d'automorphismes de graphes tel que Saucy [1]. L'élimination des symétries est implémentée dans un solveur SAT que nous avons expérimenté et comparé sur plusieurs instances SAT. Les résultats obtenus sont encourageants et montrent que l'exploitation des symétries locales donne de meilleurs résultats que l'exploitation des symétries globales sur certaines instances SAT et que la combinaison de l'exploitation des deux types de symétries sont complémentaires.

Le reste du papier est organisé comme suite : la section 2 rappelle les notions élémentaires sur le problème de satisfiabilité et les permutations. La section 3 définie le principe de la symétrie et donne quelques propriétés. La quatrième section décrit la nouvelle méthode de détection et d'élimination de la symétrie que nous proposons. La section 5 montre comment la coupure de symétrie est intégrée dans une procédure du type Davis et Putnam. Nous évaluons la méthode proposée dans la sixième section où plusieurs instances SAT sont testées et où une comparaison avec d'autres

méthodes est donnée. Finalement, nous concluons ce travail dans la section 7.

2 Quelques rappels en logique propositionnelle

2.1 La loogique propositionnelle

Nous supposons que le lecteur est familier avec le calcul propositionnel. Nous donnons ici, une courte description, une plus complète description peut être trouvée dans [21]. Soit V l'ensemble des variables propositionnelles que l'on peut appeler simplement, variables. Les variables doivent être distinguées des littéraux, qui sont en fait, les variables assignées à une des deux valeurs possibles, 1 ou 0, qui veut dire Vrai ou Faux respectivement(True ou False en Anglais). Cette distinction peut être ignorée si cela convient et ne porte pas à confusion. Pour une variable propositionnelle p, il y a deux littéraux : p le littéral positif et $\neg p$ le négatif.

Une clause est une disjonction de littéraux $\{p_1, p_2, \ldots, p_n\}$ tel qu'aucun littéral n'apparaît plus d'une fois, ni un littéral et sa négation en même temps. Cette clause est désignée par $p_1 \vee p_2 \vee \ldots \vee p_n$. Un système $\mathcal{F}$ de clauses est une conjonction de clauses. En d'autres mots, on dit que $\mathcal{F}$ est sous la forme normale conjonctive (CNF).

Un assignement qui vérifie un système de clauses $\mathcal{F}$ est une application I définit de l'ensemble des variables de $\mathcal{F}$ vers l'ensemble {Vrai, Faux}. Si $I[p]$ est la valeur du littéral positif p alors $I[\neg p] = 1 - I[p]$. La valeur de la clause $p_1 \vee p_2 \vee \ldots \vee p_n$ dans I est vraie, si la valeur vrai est assignée à au moins un de ses littéraux dans I, faux sinon. Par convention, nous définissons la valeur de la clause vide ($n = 0$) à faux.

La valeur $I[\mathcal{F}]$ du système de clauses est vrai si la valeur de chaque clause de $\mathcal{F}$ est vrai, faux, sinon. On dit que le système de clauses $\mathcal{F}$ est satisfaisable s'ils existent des assignements I qui vérifient le système de clauses et qui assignent la valeur vrai à $\mathcal{F}$, sinon il est insatisfaisable. Dans le premier cas I est appelé un modèle de $\mathcal{F}$. Notons que le système de clauses qui contient la clause vide est insatisfaisable.

On sait que [27] pour chaque formule propositionnelle $\mathcal{F}$ il existe une formule $\mathcal{F}'$ sous la forme normale conjonctive (CNF) telle que la taille de $\mathcal{F}'$ est au plus 3 fois plus longue que la formule $\mathcal{F}$ et que $\mathcal{F}'$ est satisfaisable si et seulement si $\mathcal{F}$ est satisfaisable.

Dans la suite nous allons assumer que les formules sont données sous la forme normale conjonctive.

2.2 Les permutations

Soit l'ensemble $\Omega = \{1, 2, \ldots, N\}$ pour un entier donné N, où chaque entier représente une variable propositionnelle. Une permutation de Ω est une application bijective σ de Ω à Ω qu'on représente usuellement comme un produit de cycles de permutations. On dénote par $Perm(\Omega)$ l'ensemble des toutes les permutations de Ω et $\circ$ la composition de permutations de $Perm(\Omega)$. La paire $(Perm(\Omega), \circ)$ forme le groupe de permutation de Ω. En effet, $\circ$ est close, associative, l'inverse d'une permutation est une permutation et la permutation identité 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 forme muni de l'opération $\circ$ un groupe.

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

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

3 La symétrie

Depuis la définition de symétrie de Krishnamurthy [20] dans la logique propositionnelle, plusieurs d'autres définitions ont été données par la communauté CP. Freuder dans son papier [14], a introduit les notions d'interchangeabilité globale et locale, où deux valeurs d'un domaine sont interchangeables dans un CSP, si elles peuvent être substituées l'une à l'autre sans aucun effet sur le CSP. En revanche Benhamou dans [3] a défini deux niveaux de symétrie sémantique et une notion de symétrie syntaxique. Il a également montré que l'interchangeabilité globale de Freuder est un cas particulier de symétrie sémantique et que l'interchangeabilité de voisinage (locale) est un cas particulier de la symétrie syntaxique.

Plus récemment, Cohen et al [10] ont fait un état de l'art sur les définitions de symétrie les plus connus dans les CSPs et ont les regrouper dans deux définitions : les symétries de solutions (sémantiques) et les symétries de contraintes (syntaxiques). Presque toutes ces définitions peuvent être identifiées comme appartenant à l'une des deux familles de symétrie : la symétrie syntaxique ou la symétrie sémantique. Nous allons définir dans ce qui suit les deux symétries sémantiques et syntaxiques dans la logique propositionnelle et nous allons montrer leur relation avec les symétries de so-

lutions ainsi que les symétries de contraintes dans les CSPs.

3.1 La symétrie dans la logique propositionnelle

Définition 1 (La symétrie sémantique) *Soit $\mathcal{F}$ une formule propositionnelle donnée sous la forme CNF et $L_{\mathcal{F}}$ son ensemble complet*[1] *de littéraux. Une symétrie sémantique de $\mathcal{F}$ est une permutation σ définie sur $L_{\mathcal{F}}$ telle que $\mathcal{F} \models \sigma(\mathcal{F})$ et $\sigma(\mathcal{F}) \models \mathcal{F}$.*

En d'autres mots, une symétrie sémantique d'une formule est une permutation. Nous rappelons dans la suite, la définition de la symétrie syntaxique donnée dans [4, 5].

Définition 2 (La symétrie syntaxique) *Soit $\mathcal{F}$ une formule propositionnelle donnée sous la forme CNF et $L_{\mathcal{F}}$ son ensemble complet de littéraux. Une symétrie syntaxique de $\mathcal{F}$ est une permutation σ définie sur $L_{\mathcal{F}}$ telle que la condition suivante soit vérifiée :*

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

En d'autres mots, une symétrie syntaxique d'une formule est une permutation de littéraux qui laisse invariant la formule. Si on dénote 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 des permutations de $L_{\mathcal{F}}$ qui sont les symétries syntaxiques de $\mathcal{F}$, alors $Sym(L_{\mathcal{F}})$ est trivialement un sous groupe de $Perm(L_{\mathcal{F}})$.

Remarque 1 *Les définitions de symétrie introduites dans les CSPs [10] sont liées à celles introduites dans la logique propositionnelle. On considère par exemple l'encodage SAT $\mathcal{F}$ directement du CSP P [19] où une variable booléenne est introduite pour chaque paire variable-valeur du CSP, et où une clause interdisant chaque tuple rejeté par une contrainte spécifique est ajoutée ainsi qu'une autre clause garantissant que la valeur choisie pour chaque variable est préalablement dans son domaine. Il est trivial de voir que la symétrie de solutions du CSP P est équivalente à la symétrie sémantique (la définition 1) de son encodage SAT $\mathcal{F}$ et que la symétrie de contraintes de P est équivalente à la symétrie syntaxique (la définition 2) de $\mathcal{F}$.*

Théorème 1 *Chaque symétrie syntaxique d'une formule $\mathcal{F}$ est une symétrie sémantique de $\mathcal{F}$.*

1. L'ensemble des littéraux contenant chaque littéral de $\mathcal{F}$ et son négatif.

Preuve 1 *Il est trivial de voir que la symétrie syntaxique est une condition suffisante pour la symétrie sémantique. En effet, si σ est une symétrie syntaxique de $\mathcal{F}$, alors $\sigma(\mathcal{F}) = \mathcal{F}$, donc il en résulte que $\mathcal{F}$ et $\sigma(\mathcal{F})$ ont les mêmes modèles. Chaque symétrie syntaxique est une symétrie sémantique et en général, l'inverse est n'est pas vrai.*

Exemple 1 *Soit $\mathcal{F}$ l'ensemble des clauses suivantes :*
$\mathcal{F}=\{a\vee b\vee c, \neg a\vee b, \neg b\vee c, \neg c\vee a, \neg a\vee\neg b\vee\neg c\}$ *et σ_1 et σ_2 deux permutations définies sur l'ensemble complet $L_{\mathcal{F}}$ des littéraux participant dans $\mathcal{F}$ comme suit :*
$\sigma_1=(a,b,c)(\neg a,\neg b,\neg c)$
$\sigma_2=(a,\neg a)(b,\neg c)(c,\neg b)$
Les deux σ_1 et σ_2 sont des symétries syntaxiques de $\mathcal{F}$, puisque $\sigma_1(\mathcal{F})=\mathcal{F}=\sigma_2(\mathcal{F})$.

Dans la suite, nous travaillerons uniquement sur les symétries syntaxiques, nous utiliserons donc uniquement le mot symétrie pour designer la symétrie syntaxique.

Définition 3 *Deux littéraux ℓ et ℓ' d'une formule $\mathcal{F}$ sont symétriques, s'il existe une symétrie σ de $\mathcal{F}$ telle que $\sigma(\ell) = \ell'$.*

Définition 4 *Soit $\mathcal{F}$ une formule, l'orbite d'un littéral $\ell \in L_{\mathcal{F}}$ sur lequel le groupe de symétries $Sym(L_{\mathcal{F}})$ opère est $\ell^{Sym(L_{\mathcal{F}})}=\{\sigma(\ell) : \sigma \in Sym(L_{\mathcal{F}})\}$.*

Proposition 1 *Tous les littéraux d'une orbite ℓ sont symétriques deux à deux.*

Preuve 2 *La preuve est une conséquence triviale des deux définitions précédentes.*

Exemple 2 *Dans l'exemple 1, l'orbite du littéral a est $a^{Sym(L_{\mathcal{F}})} = \{a, b, c, \neg a, \neg b, \neg c\}$. Nous pouvons voir que tous les littéraux sont dans la même orbite. Donc, ils sont tous symétriques.*

Si I est un modèle de $\mathcal{F}$ et σ une symétrie, nous pouvons avoir un autre modèle de $\mathcal{F}$ en appliquant σ sur les variables qui apparaissent dans I. Autrement dit, si I est un modèle de $\mathcal{F}$ alors $\sigma(I)$ est un modèle de $\mathcal{F}$. Une symétrie σ transforme chaque modèle en un autre modèle et chaque no-good en un autre no-good. Dans la proposition suivante, nous assumons que σ est une symétrie de l'ensemble des clauses $\mathcal{F}$.

Proposition 2 *Soit ℓ un littéral, σ une symétrie telle que $\ell' = \sigma(\ell)$ et $I' = \sigma(I)$. Si I est telle que $I[\ell] = Vrai$, alors I' est telle que $I'[\ell'] = Vrai$*

Preuve 3 *La preuve est triviale. En effet, si ℓ est vrai dans le modèle I alors $\sigma(\ell)= \ell'$ doit être vrai dans le modèle $\sigma(I)=I'$.*

Nous déduisons la proposition suivante.

Proposition 3 *Si un littéral ℓ a la valeur vrai dans un modèle de $\mathcal{F}$, alors $\sigma(\ell)$ doit avoir la valeur vrai dans un modèle de $\mathcal{F}$.*

Théorème 2 *Soient ℓ et ℓ' deux littéraux de $\mathcal{F}$ qui sont dans la même orbite en ce qui concerne le groupe de symétries $Sym(L_{\mathcal{F}})$, alors ℓ est vrai dans un modèle de $\mathcal{F}$ si et seulement si ℓ' est vrai dans un modèle de $\mathcal{F}$.*

Preuve 4 *Si ℓ est dans la même orbite que ℓ' alors il est symétrique avec ℓ' dans $\mathcal{F}$. Donc, il existe une symétrie σ de $\mathcal{F}$ tel que $\sigma(\ell) = \ell'$. Si I est un modèle de $\mathcal{F}$ alors $\sigma(I)$ est aussi un modèle de $\sigma(\mathcal{F}) = \mathcal{F}$, en plus, si $I[\ell] = true$ alors $\sigma(I[\ell']) = true$ (la proposition 2). Pour l'inverse, il faut considérer $\ell = \sigma^{-1}(\ell')$, et faire une preuve similaire.*

Corolaire 1 *Soit ℓ un littéral de $\mathcal{F}$, si ℓ n'est pas vrai dans aucun modèle de $\mathcal{F}$, alors chaque littéral $\ell' \in orbit^{L_{\mathcal{F}}}$ n'est pas vrai dans aucun modèle de $\mathcal{F}$.*

Preuve 5 *La preuve est une conséquence directe du théorème 2.*

Le corolaire 1 exprime une propriété importante que nous allons utiliser pour éliminer les symétries locales à chaque noeud de l'arbre de recherche. C'est à dire, si un échec est détecté après avoir affecté la valeur vrai au littéral courant ℓ, alors nous calculons l'orbite de ℓ et nous assignons la valeur faux à chaque littéral dans l'orbite, puisque par symétrie nous savons que la valeur vrai ne peut être assignée à ces littéraux sous peine d'avoir de produire un échec, donc les littéraux assignés à vrai de l'orbite ne participe dans aucun modèle de la sous formule considérée.

De nombreux problèmes difficiles pour la résolution ont été démontrés de complexités polynômiales si la symétrie est considérée. Par exemple, trouver des nombres de Ramsey ou résoudre le problème des pigeons sont connus pour être exponentielles pour la résolution classique, tandis que de courtes démonstrations peuvent être faites pour les deux lorsqu'on exploite la symétrie dans le système de résolution. Nous allons montrer maintenant comment détecter dynamiquement la symétrie locale.

4 La détection et l'élimination de la symétrie locale

Les symétries locales doivent être détectées dynamiquement à chaque noeud de l'arbre de recherche. La détection dynamique de la symétrie a été étudiée dans [4, 5] où une méthode de recherche des symétries syntaxiques locales a été donnée. Cependant, cette méthode n'est pas complète, elle détecte uniquement une symétrie σ à chaque noeud de l'arbre de recherche lors de l'échec dans l'affection du littéral courant ℓ. Une heuristique est utilisée sur les permutations de variables de σ à fin d'avoir le nombre maximal de littéraux dans le même cycle de permutation dans lequel apparait ℓ. En dépit de cette heuristique, cette méthode ne détecte pas tous les littéraux symétriques avec ℓ correspondant à l'orbite de ℓ, puisqu'il n'utilise pas toutes les symétries locales.

Comme alternative à cette méthode de recherche de symétrie incomplète, nous avons adapté Saucy [1] pour détecter toutes les symétries syntaxiques et nous montrons comment éliminer ces symétries durant la recherche. Saucy est un outil pour le calcul des automorphismes de graphes. Il existe d'autres outils comme Nauty [22] et plus récemment AUTOM [26] ou aussi celui décrit dans [23] qui peuvent être adaptés pour la détection des symétries locales. Il est montré dans [26] que AUTOM est l'un des meilleurs outils. Cependant, il n'est pas gratuit et comme depuis peu, une nouvelle version plus performante de Saucy vient de sortir, [13] ; nous avons choisit Saucy. Il est montré dans [12, 1, 2] que chaque formule CNF $\mathcal{F}$ peut être représentée par un graphe $G_{\mathcal{F}}$ qui est construit de la façon suivante :

- Chaque variable booléenne est représentée par deux sommets (sommet littéral) dans $G_{\mathcal{F}}$: le littéral positif et son négatif. Ces deux sommets sont connectés par une arête dans le graphe $G_{\mathcal{F}}$.
- chaque clause non binaire est représentée par un sommet (sommet clause). Une arête connecte ce sommet à chaque sommet représentant un littéral de la clause.
- Chaque clause binaire est représentée par une arête connectant les deux sommets représentant les deus littéraux de la clause. Les sommets correspondants aux clauses binaires ne sont pas ajoutés.

Une propriété importante du graphe $G_{\mathcal{F}}$ est qu'il préserve le groupe de symétries syntaxiques de $\mathcal{F}$. En effet, le groupe de symétries syntaxiques de la formule $\mathcal{F}$ est identique au groupe d'automorphismes de sa représentation graphique $G_{\mathcal{F}}$, donc nous utilisons Saucy sur $G_{\mathcal{F}}$ pour détecter le groupe de symétries syntaxiques de $\mathcal{F}$. Saucy retournes l'ensemble des générateurs *Gen* du groupe de symétries duquel on peut déduire chaque symétrie. Saucy offre la possibilité de colorer les sommets du graphe tels que, chaque sommet est autorisé à être permuté avec un autre sommet s'ils ont la même couleur. Ceci restreint les permutations aux noeuds ayant la même couleur. Deux couleurs sont utilisées dans $G_{\mathcal{F}}$, une pour les som-

mets correspondant aux clauses de $\mathcal{F}$ et l'autre couleur pour les sommets représentant les littéraux de $L_{\mathcal{F}}$. Ceci permet de distinguer les sommets clauses des sommets littéraux, et prévient donc la génération de symétries entre clauses et littéraux. Le code source de Saucy peut être trouvé à l'adresse (http ://vlsicad.eecs.umich.edu/BK/SAUCY/).

Exemple 3 *Soit $\mathcal{F}$ la formule CNF donnée dans l'exemple 1. Son graphe associé $G_{\mathcal{F}}$ est donné dans la figure 1.*

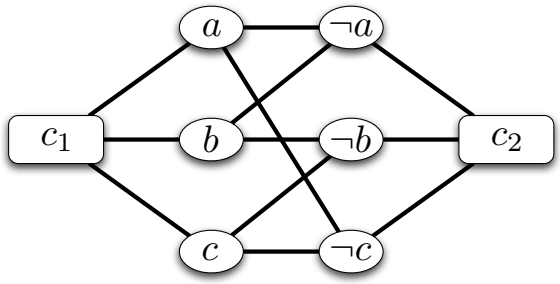

FIGURE 1 – Le graphe $G_{\mathcal{F}}$ correspondant à $\mathcal{F}$

4.0.1 Détection dynamique de la symétrie :

On considère la formule CNF $\mathcal{F}$, et un assignement partiel I de $\mathcal{F}$ où ℓ est le littéral courant en cours d'assignement. L'assignement I simplifie la formule donnée $\mathcal{F}$ en une sous formule $\mathcal{F}_I$ qui définit un état de l'espace de recherche correspondant au noeud courant n_I de l'arbre de recherche. L'idée, est de maintenir dynamiquement le graphe $G_{\mathcal{F}_I}$ de la sous formule $\mathcal{F}_I$ correspondant au sous problème local défini au noeud courant n_I, alors on colorie le graphe $G_{\mathcal{F}_I}$ et on calcule son groupe d'automorphismes $Aut(\mathcal{F}_I)$. La sous formule $\mathcal{F}_I$ peut être vu comme le sous problème restant correspondant à la partie non encore résolue. En appliquant Saucy sur ce graphe coloré, nous pouvons déduire l'ensemble des générateurs Gen du sous groupe de symétries existant entre les littéraux de $L_{\mathcal{F}_I}$ à partir desquels on peut calculer l'orbite du littéral courant ℓ qui sera utilisée pour faire des coupures de symétries.

4.0.2 Elimination des symétries :

Nous utilisons le corolaire 1 pour élaguer des espaces de recherche des méthodes de résolution. En effet, si l'assignement de la valeur vrai au littéral courant ℓ défini à un noeud donné n_I de l'arbre de recherche est montré qu'il mène à un échec, alors l'assignement de la valeur vrai à n'importe quel littéral de l'orbite de ℓ mènera aussi à un échec. Donc, la valeur $faux$ doit être assignée à chaque littéral de l'orbite de ℓ. Nous élaguons alors, le sous espace correspondant à l'assignement de la valeur alternative $vrai$ à ces littéraux dans l'arbre de recherche. C'est ce qu'on appelle les coupures de symétries.

5 Exploitation de la symétrie dans un algorithme de résolution

Maintenant nous allons montrer comment ces littéraux symétriques peuvent être utilisés pour augmenter l'efficacité des algorithmes de résolution SAT. Nous choisissons dans notre implémentation la procédure Davis Putnam (DP) comme méthode de base que nous souhaitons améliorer par l'exploitation de la symétrie.

Si I est une interprétation partielle inconsistante dans laquelle l'assignement de la valeur $vrai$ au littéral courant ℓ est montrée être en conflit, alors en accord avec le corolaire 1, tous les littéraux dans l'orbite de ℓ calculés en utilisant le groupe $Sym(\mathcal{F}_I)$ retourné par Saucy sont symétriques à ℓ. Ainsi, nous assignons la valeur faux à chaque littéral de $\ell^{Sym(L_{\mathcal{F}})}$ puisque la valeur vrai est montrée être contradictoire, et alors nous élaguons le sous espace correspondant aux assignements à la valeur vrai. La procédure résultante appelée Satisfiable est donnée dans l'algorithme 1.

La fonction $orbit(\ell, Gen)$ est élémentaire, elle permet de calculer l'orbite du littéral ℓ à partir de l'ensemble des générateurs Gen retourné par Saucy.

6 Expérimentations

Nous allons maintenant étudier les performances de notre méthode par le biais d'expérimentations. Nous choisissons pour notre étude des instances SAT pour mettre en évidence l'intérêt de l'exploitation de la symétrie dans la satisfiabilité. Nous posons que l'apport de la symétrie sera plus important dans des applications réelles. Ici, nous avons testé et comparé 4 méthodes :

1. **No-sym :** recherche sans élimination de symétries, c'est le solveur LSAT de base [24] ;
2. **Global-sym** recherche avec détection et élimination des symétries globales. Cette méthode exploite un programme nommé SHATTER, comme préprocesseur [1, 2] qui détecte et élimine les symétries globales de l'instance considérée en ajoutant à l'instance des clauses éliminant ces symétries. L'instance obtenue est alors résolu par le solveur LSAT. Le temps CPU de *Global-sym* dans la table 1 inclut le temps nécessaire à SHATTER pour le calcul et l'élimination des symétries globales.

Algorithm 1: La procédure Davis Putnam muni de l'élimination des symétries locales

Procedure: Satisfiable($\mathcal{F}$)

```
1  begin
2      if F = ∅ then F est satisfaisable
3      else si F contient la clause vide alors F est insatisfaisable
4      else begin
5          if il existe un mono-littéral ou un littéral monotone ℓ then
6          if Satisfiable(F_ℓ) then F est satisfaisable
7          else F est insatisfaisable
8          else begin
9              Choisir un littéral non assigné ℓ de F
10             if Satisfiable(F_ℓ) then F est satisfaisable
11             else begin
12                 Gen=Saucy(F);
13                 ℓ^{Sym(L_F)}=orbit(ℓ,Gen)={ℓ_1, ℓ_2, ..., ℓ_n};
14                 if Satisfiable(F_{¬ℓ_1∧¬ℓ_2∧...∧¬ℓ_n}) then F est satisfaisable else F est insatisfaisable
15             end
16         end
17     end
18 end
```

		No-sym		Global-sym		Local-sym		Global-Local-sym	
Instance	*Vars* : *clauses*	*Noeuds*	*Temps*	*Noeuds*	*Temps*	*Noeuds*	*Temps*	*Noeuds*	*Temps*
fpga10_8_SAT	120 : 448	6,637,776	44.41	449	0.02	9835	2.09	449	0.71
fpga10_9_SAT	135 : 549	-	>1,000	284	0.02	57080	20.37	284	0.53
fpga12_8_SAT	144 : 560	6,637,776	35.79	165	0.00	9835	2.14	165	0.32
fpga13_10_SAT	195 : 905	-	>1,000	4261	0.41	304,830	134.89	4261	14.08
Chnl10_11	220 : 1122	3,628,800	100.09	382	0.09	512	2.42	382	3.33
Chnl10_12_3	240 : 1344	3,628,800	120.72	322	0.10	512	2.63	322	3.41
Chnl11_12_3	264 : 1476	-	>1,000	1123	0.26	1024	6.28	1123	12.09
Chnl11_13	286 : 1742	-	>1,000	814	0.25	1024	7.38	814	10.96
Chnl11_20	440 : 4220	-	>1,000	523	0.38	1024	18.93	523	18.90
Urq3_5	46 : 470	-	>1,000	16384	0.16	30	0.09	15	0.00
Urq4_5	74 : 694	-	>1,000	-	>1,000	44	0.32	31	0.10
Urq5_5	121 : 1210	-	>1,000	-	>1,000	73	1.43	44	0.27
Urq6_5	180 : 1756	-	>1,000	-	>1,000	110	4.76	84	2.03
Urq7_5	240 : 2194	-	>1,000	-	>1,000	147	9.32	108	3.44
Urq8_5	327 : 3252	-	>1,000	-	>1,000	225	27.11	171	9.18

TABLE 1 – Quelques résultats sur des instances SAT

3. **Local-sym :** recherche avec détection et élimination des symétries locales. Cette méthode implémente dans LSAT la stratégie de détection et d'élimination dynamique décrite dans ce travail. Le temps CPU de *Local-sym* inclut le coût en temps de l'exploitation des symétries locales.
4. **Global-Local-sym :** recherche qui combine l'exploitation des symétries globales et locales. La méthode consiste à utiliser LSAT avec exploitation des symétries locale (à savoir donc la méthode *Local-sym*) sur les instances produites par l'utilisation de SHATTER comme préprocesseur.

sur de différentes instances SAT comme les FPGA (Field Programmable Gate Array), Chnl, Urquhart et quelques instances issus du problème de coloration de graphes. Nous rappelons que le point commun entre les différentes méthodes présentées précédemment est qu'elles ont toutes en commun la méthode LSAT comme méthode de base. Les indicateurs de complexité sont le nombre de noeuds ainsi que le temps (en secondes). Le temps nécessaire pour le calcul et l'exploitation des symétries globales et locales est ajouté au temps CPU total pour la résolution des instances. Le code source est en C, il est compilé et exécuté sur une machine équipée d'un Pentium 4, 2.8 GHZ et 1 Gb de RAM.

6.1 Résultats sur les instances SAT

La table 1 montre les premiers résultats obtenus des différentes méthodes sur les instances SAT choisit. Elle donne le nom des instances, la taille de celles-ci (*variables*/*clauses*), le nombre de noeuds et le temps CPU pour la résolution des instances pour chaque méthode.

La table 1 montre que *Global-sym* est en général meilleure que *Local-sym* et *No-sym* en nombre de noeuds et en temps CPU sur les problèmes $FPGA$ et $Chnl$, mais *Local-sym* est capable de les résoudre aussi. Ces problèmes contiennent un nombre très important de symétries globales, ceci explique qu'il est suffisant de les éliminer afin de résoudre efficacement ces instances. Eliminer les symétries locales sur ces problèmes peut parfois rendre la résolution plus lente. Les instances Urq sont connus pour être plus durs que les $FPGA$ et les $Chnl$, nous pouvons voir que *No-sym* n'est pas capable de les résoudre et que *Global-sym* n'a résolu que l'instance $Urq3\_5$ et n'a pas pu résoudre les autres dans la limite de temps imposée. La méthode *Local-sym* a résolu tous les instances Urq efficacement. L'élimination de la symétrie locale dans ce cas est plus avantageuse que l'élimination des symétries globales. Nous pouvons voir qu'en moyenne la méthode *Global-Local-sym* est meilleure que toutes les autres méthodes, puisqu'elle a résolu toutes les instances efficacement. Ses performances son comparable à la méthode *Global-sym* sur les instances $FPGA$ et $Chnl$ et à la méthode *Local-sym* sur les instances Urq. Il est donc plus avantageux de combiner les deux types d'élimination de la symétrie pour ces problèmes. Les résultats confirment que les deux méthodes *Global-sym* et *Local-sym* peuvent être complémentaires.

6.2 Les résultats sur les instances du problème de coloration de graphes

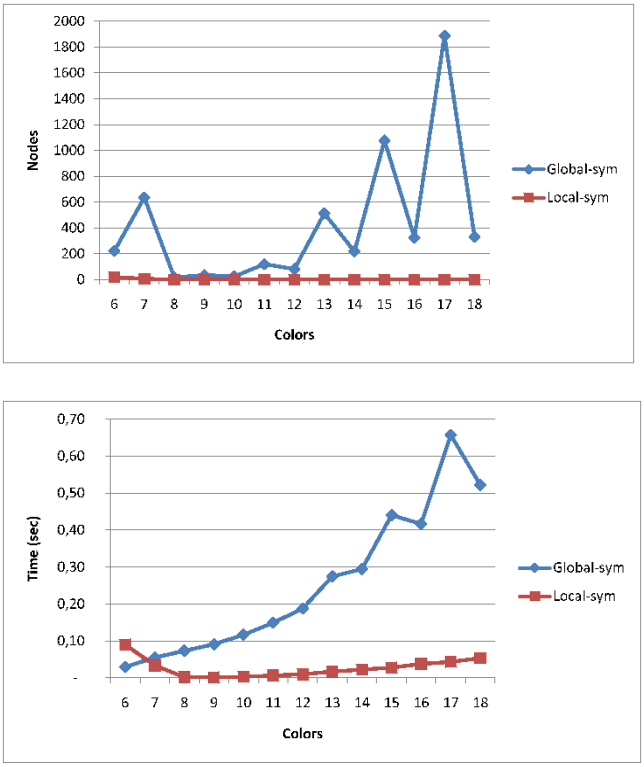

FIGURE 2 – Les courbes de noeuds et temps en moyenne pour les deux méthodes d'élimination de la symétrie sur le problème de coloration de graphes où $n = 30$ et $d = 0.5$

Les instances du problème de coloration de graphes sont générées en fonction des paramètres suivants : (1) n : le nombre de sommets, (2) $Colors$: le nombre de couleurs et (3) d : la densité qui est un nombre entre 0 et 1 qui exprime le ratio : le nombre de contraintes (le nombre d'arêtes dans le graphe) sur le nombre de toutes le contraintes possibles. Pour chaque test correspondant aux paramètres n, $Colors$ et d fixés, un échantillon de 100 instances est générée aléatoirement et les mesures (temps CPU, nombre de noeuds) sont prises en moyenne.

Nous avons reporté sur la figure 2 les résultats ob-

tenus à partir des méthodes testées : *Global-sym*, et *Local-sym*, sur les instances générées aléatoirement où nous avons fixé le nombre de variables à $n = 30$ et où la densité est fixée à $(d = 0.5)$. Les courbes donnent le nombre moyen de noeuds, respectivement, le temps CPU moyen en fonction du nombre de couleur pour chaque méthode.

Nous pouvons voir que les courbes représentant le nombre moyen de noeuds (les courbes du haut) que la méthode *Local-sym* détecte et élimine plus de symétries que la méthode *Global-sym* et que *Global-sym* n'est pas stable pour le problème de coloration de graphe. A partir des courbes de temps (les courbes du bas), nous pouvons voir que *Local-sym* est en moyenne plus rapide que *Global-sym* bien que Saucy est exécuté à chaque noeud pour la méthode *Local-sym*. L'élimination des symétries locale est plus avantageuse pour la résolution du problème de coloration de graphes que l'élimination des seules symétries globales sur ces problèmes.

7 Conclusion et perspectives

Ici, nous avons étendu le principe de détection et d'élimination des symétries aux symétries locales. En effet, les symétries de chaque sous formule CNF définie à chaque noeud de l'arbre de recherche et qui est dérivée de la formule initiale en considérant l'assignement partiel correspondant à ce noeud. Nous avons adapté Saucy pour calculer ces symétries locales en maintenant dynamiquement le graphe correspondant la sous formule définie à chaque noeud de l'arbre de recherche.

On donne à Saucy en entrée le graphe de la sous formule locale et il nous retourne alors l'ensemble des générateurs du groupe d'automorphismes du graphe donné, sachant que ce groupe est équivalent au groupe de symétries locale de la sous formule considérée. La technique de détection et d'élimination des symétries locales a été implémentée dans une méthode de résolution de problèmes SAT appelée *LSAT* afin d'améliorer ses performances. Les résultats expérimentaux confirment que l'élimination des symétries locales est d'un apport non négligeable pour la résolution des problèmes SAT et améliore les méthodes n'exploitant que l'élimination des symétries globales sur certains problèmes, et qu'elle peut être complémentaire à l'élimination des symétries globales si on les combine.

Comme travail futur, nous envisageons d'implémenter une version affaiblie des conditions de détection des symétries locales que nous supposons plus avantageuse pour détecter un plus grand nombre de symétrie, que nous comparerons ses résultats avec ceux qu'on vient de présenter.

Un autre point important est de tenter de détecter les symétries de variables locales et de poster dynamiquement des contraintes qui les éliminent. Il serait alors important de comparer les approches statiques qui détectent uniquement les symétries globales avec cette approche.

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] 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.

[4] 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.

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

[6] 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.

[7] Belaïd Benhamou and Mohamed Réda Saïdi. Local symmetry breaking during search in csps. In Springer, editor, *The 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 195–209, Providence, USA, 2007.

[8] W. Bibel. Short proofs of the pigeon hole formulas based on the connection method. *Automated reasoning*, (6) :287–297, 1990.

[9] Brown, C. A.Finkelstein, and L. P. W. Purdom. Bactrack searching in the presence of symmetry. *In t. Mora (ed), Applied algebra, algebraic algorithms and error-correcting codes, 6th International Conference. Springer-Verlag*, (6) :99–110, 1988.

[10] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *In, proceedings of CP*, pages 17–31, 2005.

[11] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. *Workshop on Tractable Resonning, AAAI-92, San Jose*, July 1992.

[12] 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.

[13] P. T. Darga, K. A. Sakallah, and I. L. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45st Design Automation Conference, Anaheim, California*, 2008.

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

[15] Ian P. Gent, Tom Kelsey, S. A. Linton, J. Pearson, and Colva M. Roney-Dougal. Groupoids and conditional symmetry. In *CP*, pages 823–830, 2007.

[16] J. W. L. Glaisher. On the problem of the eight queens. *Philosophical Magazine*, 48(4) :457–467, 1874.

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

[18] J.G. Kalbfleisch and R.G. Stanton. On the maximal triangle-free edge-chromatic graphs in three colors. *combinatorial theory*, (5) :9–20, 1969.

[19] J. De Kleer. A comparison of atms and csp techniques. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 290–296, 1989.

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

[21] R.C. Lyndon. *Notes of logic*. Van Nostrand Mathematical Studies, 1964.

[22] B McKay. Practical graph isomorphism. In *Congr. Numer. 30*, pages 45–87, 1981.

[23] C. Mears, M. Garcia de la Banda, and M. Wallace. On implementing symmetry detection. In *The CP 2006 Workshop on Symmetry and Constraint Satisfaction Problems (SymCon'06)*, pages 1–8, Cité des Congrès - Nantes, France, septembre 2006.

[24] R. Ostrowski, B. Mazure, and L. Sais. Lsat solver. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, 2002.

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

[26] J. F. Puget. Automatic detection of variable and value symmetries. In LNCS Springer, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-2005)*, pages 474–488, Sitges, Spain, october 2005.

[27] P. Siegel. Representation et utilisation de la connaissance en calcul propositionnel, 1987. Thèse d'état, GIA - Luminy (Marseille).