Élimination des symétries dans une nouvelle méthode de recherche de modèles stables

Tarek Khaled, Belaid Benhamou

Élimination des symétries dans une nouvelle méthode de recherche de modèles stables

Tarek Khaled * **Belaïd Benhamou**

Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France.
{tarek.khaled,belaid.benhamou}@univ-amu.fr

Abstract

La notion de symétrie est largement étudiée dans différents domaines, en particulier dans la programmation par contraintes où plusieurs travaux ont été réalisés. L'élimination des symétries a permis d'améliorer de manière significative les performances de nombreux solveurs. Généralement, les problèmes combinatoires contiennent un nombre important de symétries rendant leurs résolutions impraticables par les solveurs qui ne les considèrent pas. En effet, ces symétries guident souvent les solveurs à explorer inutilement des branches symétriques et redondantes. Dans ce papier, nous nous sommes intéressé à la programmation par ensembles réponses (Answer Set Programming, ASP). L'ASP est un paradigme bien connu dans la représentation et le raisonnement sur les connaissances. Cependant, seuls quelques travaux concernant l'exploitation des symétries dans l'ASP ont été faits. Dans ce travail, nous étudions la détection et l'élimination des symétries pour une nouvelle méthode ASP que nous introduisons. Cette méthode est basée sur une nouvelle sémantique qui étend celles des modèles stables. Pour montrer l'impact de l'élimination des symétries dans notre méthode, nous l'avons expérimenté avec et sans symétries sur une grande variété de problèmes combinatoires et nous l'avons comparé à d'autres systèmes ASP connus. Les résultats obtenus sont très prometteurs.

1 Introduction

La programmation par ensembles réponses (ASP) est un important paradigme utilisé pour la représentation des connaissances et le raisonnement non-monotone. Cette approche est utilisé dans une variété de problèmes hautement combinatoires, parmi eux les problèmes de graphe, de planification et le modèle checking. L'ASP est appliqué dans la robotique, la bio-informatique mais aussi dans l'industrie. C'est un outil de modélisation très expressif capable de représenter un nombre important de problèmes. C'est devenu une approche très populaire du fait de la disponibilité de nombreux systèmes performants comme $Clasp$[13], DLV [18] et $Smodels$ [25] et d'autres systèmes basé sur les solveurs SAT comme $ASSAT$ [21] et $Cmodels$ [19].

Le paradigme ASP a émergé de la recherche sur la représentation des connaissances et le raisonnement nonmonotone. De nombreuses recherches ont été effectuées pour définir la sémantique des programmes logiques et l'objectif principal est de donner un sens précis à la négation par échec. La première sémantique avait été proposée par Clark [7]. Cette sémantique est utilisée dans plusieurs solveurs ASP, mais la sémantique la plus connue pour l'ASP est celle des modèles stables [15]. Dernièrement, une nouvelle sémantique [6] a été proposé. Dans cette sémantique, un programme logique est représenté par un ensemble de clauses de Horn ayant la même taille que le programme d'entrée. L'avantage de cette sémantique est la caractérisation facile des modèles stables et l'extension qu'elle fournit pour la sémantique des modèles stables [15].

Nous proposons une nouvelle méthode pour la recherche de modèles stables qui repose sur un processus d'énumération booléenne défini pour le paradigme ASP selon la sémantique introduite dans [6]. Cette méthode a l'avantage d'effectuer le processus énumératif uniquement sur une restriction de littéraux représentant le strong backdoor (STB) du programme logique considéré [26]. La méthode cherche toutes les extensions possibles de la représentation clausale du programme logique et à partir desquelles on peut générer tous les modèles stables. Il pourrait également produire des extra-modèles qui correspondent à des extensions supplémentaires qui ne sont pas capturées par la sémantique des modèles stables. Ces extra-modèles permettent d'étendre la sémantique des modèles stables. Afin

*Papier doctorant : Tarek Khaled est auteur principal.

de comparer nos résultats à d'autres systèmes ASP, nous avons limité la recherche aux seuls modèles stables. Les modèles stables sont déduits d'un sous-ensemble d'extensions satisfaisant ce que nous appellons ici *la condition discriminante*.

D'autre part, la symétrie est étudiée dans plusieurs domaines, comme les mathématiques et l'intelligence artificielle. On dit qu'un objet est symétrique, lorsque la permutation de ses éléments laisse l'objet inchangé. La symétrie est une notion fondamentale dans les problèmes de satisfiabilité, elle permet de réduire la complexité de résolution d'un certain nombre de problèmes combinatoires. Le principe de la symétrie dans la logique propositionnelle a d'abord été présenté par Krishnamurthy dans [17]. La symétrie a été étudiée en profondeur dans [3, 4] où une élimination de symétrie dynamique est proposée pour la méthode DPLL [10] lors de la résolution du problème de satisfiabilité. Une approche statique qui élimine les symétries dans une phase de prétraitement a été introduite dans [8]. Cette approche statique consiste à ajouter des contraintes exprimant la symétrie dans la représentation initiale du problème de satisfiabilité. Cette technique a été améliorée dans [1].

Beaucoup de problèmes combinatoires exprimés dans le paradigme ASP contiennent un grand nombre de symétries. Par exemple, le problème des Pigeons est connu pour exiger un temps exponentiel de résolution lorsque les symétries ne sont pas éliminées. En effet, le solveur ASP explore tous les espaces de recherche symétriques. Il est possible d'éviter d'explorer ces espaces de recherche redondants en éliminant les symétries existantes. Jusqu'à présent, seuls quelques travaux sur l'élimination des symétries dans l'ASP ont été faits. Par exemple, la méthode présentée dans [11] traite les symétries dans l'ASP avec une approche statique et où l'encodage est basé sur la représentation corps-atomes d'un programme logique. Une autre approche est étudiée dans [5]. Cette dernière méthode élimine les symétries à la fois de façon statique et dynamique.

Dans cet article, nous discutons d'abord la nouvelle méthode ASP, puis nous traitons de la détection et de l'élimination des symétries dans la représentation en clauses de Horn que la nouvelle sémantique [6] utilise pour exprimer des programmes logiques. Nous divisons le problème de l'élimination des symétries en trois parties. Nous commençons par créer un graphe coloré qui représente la forme clausale du programme logique donné de telle sorte que les automorphismes du graphe soient identiques aux symétries de la représentation clausale. Ensuite, un ensemble de générateurs représentant le groupe d'automorphismes du graphe est calculé en utilisant des outils tel que $saucy$ [1], $autom$ [24] ou $nauty$ [22]. Enfin, des contraintes d'élimination de symétrie (symmetry-breaking predicates, SBP) sont construites et ajoutées à la formulation clausale initiale. Notre approche s'inspire des travaux présentés dans [1, 2].

Le reste de l'article est organisé comme suit. Tout d'abord, nous rappelons dans la section 2 quelques notions sur la programmation logique et la sémantique [6] sur laquelle repose notre méthode de recherche de modèles stables. Nous décrivons ensuite dans la section 3 la nouvelle méthode de recherche de modèles stables. Dans la section 4, nous donnons les définition et les propriétés théoriques de la symétrie dans cette nouvelle représentation. Nous décrivons dans la section 5 la méthode de détection des symétries avant de montrer l'approche de l'élimination de ces dernières dans la section 6. La section 7 donne les résultats expérimentaux obtenus sur certains problèmes combinatoires. La section 8 conclut le travail et donne quelques perspectives de rcherche.

2 Préliminaires

Nous présentons dans ce qui suit des notions sur la permutation, la programmation par ensembles réponses et les principales bases théoriques de la sémantique utilisée [6].

2.1 Permutations

Une permutation d'un ensemble Ω est une bijection définie de Ω vers lui même. Notons $Perm(\Omega)$ l'ensemble de toutes les permutations de Ω. La paire $(Perm(\Omega), \star)$ où $\star$ est une loi de composition de la permutation de $Perm(\Omega)$, forme le groupe de permutation de Ω. Ce groupe est représenté par l'ensemble générateur $(Gen(\Omega), \star)$. Gen est un sous-ensemble de $Perm$ et chaque élément de $Perm$ peut être écrit comme une composition d'éléments de Gen. L'orbite d'un élément ω sur lequel le groupe $Perm(\Omega)$ est appliqué $\omega^{Perm(\Omega)} = \{\omega^\sigma : \omega^\sigma = \sigma(\omega), \sigma \in Perm(\Omega)\}$.

2.2 Programmes logiques et programmation par ensemble réponses (ASP)

Un programme logique général π est un ensemble de règles de la forme *r :tête(r)* $\leftarrow$ *corps(r)*. En général, le programme est donnée dans la logique du première ordre et des systèmes "grounders" comme $gringo$ [14] ou $lparse$ sont utilisé pour produire un programme propositionnel équivalent. Un programme logique général π est un ensemble de règles de la forme : $r = A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ où $A_{i \in \{0...n\}}$ est un atome et not le symbole exprimant la négation par échec. Le corps positif de r est $corps^+(r) = \{A_1, A_2, ..., A_m\}$ et le négatif est $corps^-(r) = \{A_{m+1}, ..., A_n\}$. La projection positive de r est $r^+ = A_0 \leftarrow A_1, A_2, ..., A_m$. La signification intuitive de la règle r est la suivante : si on prouve tous les atomes de $corps^+(r)$, et qu'on n'arrive à prouver aucun des atomes de $corps^-(r)$, alors on infère A_0. Le réduit d'un programme π par rapport à un ensemble d'atomes X est le

programme positif π^X obtenu à partir de π en supprimant chaque instance de règle contenant un atome $not\ A_i$ dans son corps négatif tel que $A_i \in X$ et tous les atomes $not\ A_j$ tels que $A_j \notin X$ dans les corps négatifs des règles restantes. Formellement, $\pi^X = \{r^+ : corps^-(r) \cap X = \emptyset\}$. Dans le reste de l'article, nous nous concentrerons sur les programmes logiques généraux.

La sémantique la plus connue pour les programmes logiques généraux est celle des modèles stables [15]. Un ensemble X d'atomes est un modèle stable de π ssi X est identique au modèle minimal d'Herbrand du réduit π^X obtenu à partir de π en considérant l'ensemble des atomes X. Ce modèle est aussi appelé le modèle canonique de π^X, il est dénoté par $Cn(\pi^X)$. Formellement, un ensemble X d'atomes est un modèle stable de π ssi $X = Cn(\pi^X)$.

En pratique, plusieurs solveurs ASP sont basés sur la complétion de Clark [7]. Parmi eux, les solveurs ASP basés sur la procédure DPLL et les solveurs SAT. Pour traiter le concept de négation par échec, Clark a proposé le concept de complétion pour les programmes logiques (notation $comp(\pi)$). Il est connu que chaque modèle stable de π est un modèle de la complétion mais l'inverse n'est vrai que si le programme est sans boucles [12]. Afin d'établir l'équivalence entre les modèles stables d'un programme logique et ceux de sa complétion, des formules de boucles doivent être en général ajoutées à la complétion [21]. Mais, le nombre de formules de boucles pourrait varier d'une façon exponentielle par rapport à la taille du programme logique donnée. Par conséquent, la complexité spatiale des solveurs ASP adoptant cette approche n'est pas constante, elle pourrait varier exponentiellement dans le pire des cas [20].

2.3 La sémantique utilisée dans notre méthode

Une nouvelle sémantique est proposée dans [6]. Cette sémantique utilise une représentation sous la forme d'un ensemble de clauses de Horn pour exprimer le programme logique considéré. Cette représentation a l'avantage d'avoir la même taille que celle du programme logique d'entrée.

la nouvelle sémantique est basée sur un langage propositionnel classique L ayant deux types de variables : un sous-ensemble de variables classiques $V = \{A_i : A_i \in L\}$ et un autre $nV = \{not\ A_i : not\ A_i \in L\}$. Pour chaque variable $A_i \in V$, il existe une variable correspondante $not\ A_i \in nV$ désignant la négation par échec de A_i. La nouvelle sémantique donne un lien entre les deux types de variables (celles de V et celles de nV). Ce lien est exprimé par l'ajout au langage propositionnel L d'un axiome exprimant l'exclusion mutuelle entre chaque littéral $A_i \in V$ et son littéral négatif correspondant $not\ A_i \in nV$. Cet axiome d'exclusion mutuelle est exprimé par l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$.

Un programme logique $\pi = \{r : A_0 \leftarrow A_1, A_2, ..., A_m, not\ A_{m+1}, ..., not\ A_n\}$ où $(0 \leq m < n)$ est exprimé dans le langage propositionnel L par un ensemble de clauses de Horn propositionnelles $CR = \{A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n\}$ où $(0 \leq m < n)$ qui représente l'ensemble des règles du programme logique. Chaque règle $r \in \pi$ est traduite par une clause de Horn. Pour compléter la représentation du programme π dans cette nouvelle sémantique, on rajoute à l'ensemble des clauses exprimant les règles, l'ensemble de clauses $ME = \{(\neg A_i \vee \neg not\ A_i) : A_i \in V\}$ exprimant l'axiome d'exclusion mutuelle. La représentation logique du programme π dans le langage propositionnel L est donnée par l'ensemble de clauses $CR \cup ME$. Un programme logique est donc exprimé par un ensemble de clauses de Horn :

$$HC(\pi) = \{ \bigcup_{r \in \pi} (A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\ A_{m+1}, ..., \neg not\ A_n) \bigcup_{A_i \in V} (\neg A_i \vee \neg not\ A_i)\}.$$

L'ensemble ME peut être implémenté comme deux règles d'inférences. Il n'est plus nécessaire de mémoriser ME dans $HC(\pi)$. La représentation $HC(\pi)$ aura la même taille que le programme logique π. L'algorithme que nous allons introduire dans la section suivante travaille sur la représentation $HC(\pi)$. C'est un algorithme énumératif qui effectue l'énumération sur un sous-ensemble de littéraux qui représentent le strong backdoor (STB) [26] du programme logique d'entrée. L'ensemble STB est formé par les littéraux de la forme $not\ A_i$ qui apparaissent dans le programme logique π. Il est défini par $STB = \{not\ A_i : \exists r \in \pi, not\ A_i \in corps^-(r) \subseteq nV\}$. La méthode calcule principalement les extensions de $HC(\pi)$ qui encode les modèles stables. Étant donné un programme π et son ensemble STB, une extension de $HC(\pi)$ par rapport au STB (ou simplement une extension de la paire $(HC(\pi), STB)$ est l'ensemble des clauses consistantes dérivées de $HC(\pi)$ quand on ajoute un ensemble maximal de littéraux $not\ A_i \in STB$. Formellement :

Définition 1 *Soit $HC(\pi)$ le codage CNF d'un programme logique π, STB son strong backdoor et un sous-ensemble $S' \subseteq STB$, l'ensemble $\mathrm{E} = \mathrm{HC}(\pi) \cup \mathrm{S}'$ de clauses est alors une extension de $(\mathrm{HC}(\pi), \mathrm{STB})$ si les conditions suivantes sont vérifiées :*

1. E *est consistant,*
2. $\forall \mathrm{not\ A_i} \in \mathrm{STB} - \mathrm{S}', \mathrm{E} \cup \{\mathrm{not\ A_i}\}$ *est inconsistant.*

Il a été démontré dans [6] que chaque modèle stable d'un programme logique π est représenté par une extension E de sa forme logique $HC(\pi)$ qui vérifie la condition $(\forall \mathrm{A_i} \in \mathrm{V}, \mathrm{E} \models \neg \mathrm{not\ A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i})$. Cette condition est appelée condition discriminante dans [6]. Formellement, nous avons :

Théorème 1 *Si X est un modèle stable d'un programme logique π, alors il existe une extension E de $(HC(\pi), STB)$ telle que $\mathrm{X} = \{\mathrm{A_i} \in \mathrm{V} : \mathrm{E} \models \mathrm{A_i}\}$. D'autre part, E vérifie la condition dite discriminante : $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$.*

Exemple 1 *On considère le programme logique : $\pi = \{\ a \leftarrow not\, b\,;\, b \leftarrow not\, a\ \}$ La représentation clausale du programme logique π est composé par l'ensemble $HC(\pi) = CR \cup ME$ où $CR = \{a \vee \neg not\, b, b \vee \neg not\, a\}$, $ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b\}$ et son strong backdoor est donné par $STB = \{not\, a, not\, b\}$. Nous pouvons voir que $(HC(\pi), STB)$ admet deux extensions $E_1 = HC(\pi) \cup \{not\, a\}$ et $E_2 = HC(\pi) \cup \{not\, b\}$. En effet, E_1 and E_2 sont maximalement consistant par rapport au STB. Ces deux extensions vérifie la condition discriminante. Alors, le programme logique a deux modèles stables $M_1 = \{b\}$ et $M_2 = \{a\}$tels que $E_1 \models M_1$ et $E_2 \models M_2$.*

3 Présentation de la nouvelle méthode de recherche de modèles stables

Nous décrivons ici la nouvelle méthode de recherche pour les modèles stables basé sur la nouvelle sémantique présenté précédemment [6]. Une description préliminaire de cette méthode a été présentée dans [16]. Pour un programme logique donné π, cette méthode calcule toutes les extensions de $(HC(\pi), STB)$ à partir desquelles les modèles stables sont déduits par résolution unitaire. Intuitivement, la recherche des extensions de $(HC(\pi), STB)$ se fait par l'addition progressive des littéraux $not\, A_i$ de l'ensemble STB à $HC(\pi)$ et en vérifiant la consistance de l'ensemble obtenu à chaque nœud. Si nous nous concentrons uniquement sur les modèles stables, il suffit de ne considérer que les extensions vérifiant la condition discriminante. En d'autres termes, nous effectuons des coupures dans l'arbre de recherche pour ignorer les extensions qui ne vérifient pas cette condition.

Le processus d'énumération construit de façon incrémentale une extension en alternant dans l'arbre de recherche entre les nœuds déterministes correspondant aux propagations unitaires et les nœuds non déterministes qui représentent les points de choix. Les points de choix sont définis par l'affectation des valeurs de vérité (vrai ou faux) à certains littéraux de l'ensemble STB. Des règles d'inférence sont utilisées par la méthode pour augmenter le nombre de propagations unitaires et réduire ainsi l'espace de recherche. La méthode proposée calcule tous les modèles stables d'un programme logique donné et, en général, pourrait rechercher certains extra-modèles lorsque des modèles stables n'existent pas. Mais, dans ce travail, nous avons limité la recherche qu'aux modèles stables.

3.1 Fondements théoriques de la méthode

Nous allons maintenant introduire quelques règles d'inférence que la méthode utilisera par la suite dans le processus d'énumération de modèles stables.

Définition 2 *Soit un programme π et $HC(\pi)$ sa forme clausale. On définit sur $HC(\pi)$ les deux règles d'inférence suivantes : $\dfrac{A_i}{\neg not\, A_i}$ et $\dfrac{not\, A_i}{\neg A_i}$.*

Ces deux règles d'inférence sont une implémentation efficace de l'ensemble de clauses $ME = \bigcup_{A_i \in V} \{(\neg A_i \vee \neg not\, A_i)\}$ de $HC(\pi)$ qui exprime l'exclusion mutuel entre la paire d'atomes A_i et $\neg not\, A_i$.

Dans le cas de notre méthode, l'énumération est effectuée uniquement sur un sous-ensemble de variables STB. Soit $C_{STB} = \{c_i = \neg not\, A_{i_1} \vee, ..., \vee \neg not\, A_{i_k} / \ |\ c_i\ | \geq 1, \forall j \in \{1..k\}, not\, A_{i_j} \in STB\}$ l'ensemble de toutes les clauses négatives possibles formées par certains littéraux de l'ensemble STB. Le traitement non déterministe d'un point de choix correspondant à $not\, A_j$ est fait d'abord par son affectation à la valeur $Vrai$ pour favoriser la maximalité d'extension courante. L'exploration de la branche correspondant à l'affectation de la valeur de vérité $Faux$ à $not\, A_j$ n'est nécessaire que lorsque la première branche produit au moins une sous-clause $c_i \in C_{STB}$. Cette propriété pourrait considérablement réduire la complexité en temps de la méthode étudiée, nous la prouverons dans la proposition 1.

Proposition 1 *Soient π un programme logique, $HC(\pi)$ sa forme Horn clausale, $HC(\pi)_I$ sa forme Horn clausale simplifiée par l'instanciation partielle I correspondant au nœud courant n de l'arbre de recherche, $STB = \{not\, A_i : \exists r \in \pi, not\, A_i \in r^-\}$ son strong backdoor, et C_{STB} l'ensemble de clauses négatives possibles construites sur les littéraux de l'ensemble STB. Si $not\, A_j \in STB$ est le littéral courant à affecter au nœud n et que $\forall c_i \in C_{STB}$, $HC(\pi)_I \wedge not\, A_j \not\models c_i$, alors toute extension de $HC(\pi)_I \wedge \neg not\, A_j$ est aussi une extension de $HC(\pi)_I \wedge not\, A_j$.*

Preuve 1 *Le sous-ensemble de clauses $HC(\pi)_I$ est le système de clauses simplifié, obtenu à partir de $HC(\pi)$ par la considération des littéraux interprétés dans l'instanciation partielle I. L'ensemble de clauses $HC(\pi)_I$ représente le sous-problème correspondant au nœud courant n de l'arbre de recherche. Par hypothèse $not\, A_j$ est le prochain littéral du STB à affecter en ce point n de l'arbre. Le système de clauses simplifié $HC(\pi)_I$ correspondant au nœud n contient deux sortes de clauses : le sous ensemble de clauses de la forme $\neg not\, A_j \vee C_1$ contenant le littéral $\neg not\, A_j$ et où C_1 représente un ensemble de bouts de clauses, et le sous ensemble clauses de C_2 ne contenant pas le littéral $\neg not\, A_j$. Soit $e = not\, A_{i_1} \wedge ... \wedge not\, A_{i_k}$,*

avec $not\ Ai_j \in STB$ *une extension de* $HC(\pi)_I \wedge \neg not\ A_j$*, montrons que* e *est aussi une extension de* $HC(\pi)_I \wedge not\ A_j$*. On peut remarquer que* $HC(\pi)_I \wedge \neg not\ A_j \equiv C_2$ *et que* $HC(\pi)_I \wedge not\ A_j \equiv C_1 \wedge C_2$*. L'ensemble* e *est une extension de* $HC(\pi)_I \wedge \neg not\ A_j$*, donc* $C_2 \wedge e$ *est consistant. Pour montrer que* e *est aussi une extension de* $HC(\pi)_I \wedge not\ A_j$*, il suffit de montrer que* $C_1 \wedge C_2 \wedge e$ *est consistant. Procédons par l'absurde, en supposant que* $C_1 \wedge C_2 \wedge e$ *est inconsistant. Il en résulte que* $C_1 \wedge C_2 \wedge e \models \square$ *et donc* $C_1 \wedge C_2 \models \neg e$*. Ce qui veut dire que* $C_1 \wedge C_2 \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$*. Il en résulte que* $HC(\pi)_I \wedge not\ A_j \models \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$*. Donc* $HC(\pi)_I \wedge not\ A_j \models c_i \in C_{STB}$ *et cela contredit l'hypothèse.*

En d'autre mots, si aucune clause $c_i \in C_{STB}$ n'a été produite en un point de choix de l'arbre où on a interprété à vrai un littéral $not\ A_j \in STB$, il est alors inutile d'explorer la branche correspondant au littéral négatif $\neg not\ A_j$. Cela éviterait à la méthode d'explorer des branches redondantes et inutiles . En conséquence, cette propriété permet de réduire le nombre de points de choix de l'arbre de recherche.

Proposition 2 *Si* $HC(\pi)$ *est la forme clausale d'un programme logique* π *et* I *l'instanciation partielle courante, alors la résolution unitaire est suffisante pour produire toute clause* $c_i = \neg not\ A_{i_1} \vee ... \vee \neg not\ Ai_k \in C_{STB}$ *à partir de* $HC(\pi)_I$*.*

Preuve 2 *D'après le théorème de déduction automatique, montrer que* $HC(\pi)_I \models c_i$ *est équivalent à* $HC(\pi)_I \wedge \neg c_i \models \perp$*. Comme* $HC(\pi)$ *est à la base un ensemble de clauses de Horn, il en résulte que l'ensemble de clauses simplifié* $HC(\pi)_I \wedge \neg c_i$ *l'est aussi car* $HC(\pi)_I \wedge not\ A_{i_1} \wedge ... \wedge not\ Ai_k$ *est trivialement un ensemble de clauses de Horn. Comme la résolution unitaire est suffisante pour décider la consistance de tout ensemble de clauses de Horn, alors elle est en particulier pour* $HC(\pi)_I \wedge \neg c_i$*. En d'autres mots, la résolution unitaire est suffisante pour montrer* $HC(\pi)_I \wedge \neg c_i \models \perp$ *et, par conséquent, suffisante pour montrer* $HC(\pi)_I \models c_i$*.*

Pour appliquer la coupure induite par la proposition 1 en un point de choix donné de l'arbre de recherche, notre méthode doit prouver qu'aucune clause $c_i \in C_{STB}$ n'est produite en ce nœud. Pour ce faire, la méthode essaye de produire une telle clause en utilisant la résolution unitaire (Proposition 2).

3.2 Description de l'algorithmique

Nous présentons dans ce qui va suivre le nouvel algorithme de recherche de modèles stables. Le processus d'énumération de ce dernier est basé sur la procédure DPLL que nous avons adaptée au cas des ASP traités par la nouvelle sémantique [6]. Nous avons aussi introduit et implémenté plusieurs nouvelles règles d'inférences pour booster la méthode. Dans ce nouvel algorithme, la recherche de modèles stables alterne des phases de propagation unitaire déterministes et des phases de points de choix non déterministes où le processus de production de clauses $c_i \in C_{STB}$ est lancé sur la première branche du point de choix où un littéral $not\ A_i$ du STB est interprété à vrai. Le processus de production est inutile sur la deuxième branche du point de choix où le littéral $not\ A_i$ du STB est interprété à faux. Durant les deux phases alternées, l'algorithme affecte des valeurs de vérité aux littéraux à la manière DPLL. Si un conflit est rencontré au cours de la recherche, alors l'algorithme explore la branche correspondant à la deuxième valeur de vérité de la variable représentant le point de choix courant uniquement si une clause $c_i \in C_{STB}$ est produite. Sinon, un rebroussement (backtrack) est effectué.

Une extension est trouvée quand toutes les clauses sont satisfaites ou bien quand tous les littéraux du STB ont été tous affectés sans falsifier aucune clause. Dans les deux cas, l'algorithme exécute une phase dite de "complétion". Dans le premier cas, la méthode complète l'interprétation courante par l'assignation à vrai de l'ensemble des variables $not\ A_i$ restant du STB et par l'assignation à faux de toutes les autres variables non encore affectés (hypothèse du monde clos). Dans le deuxième cas, la complétion consiste selon l'hypothèse du monde clos, à mettre à faux les variables non affectées. Dans les deux cas, un modèle minimal candidat est trouvé. L'algorithme vérifie alors si le modèle candidat est bien un modèle stable.

L'algorithme commence par un premier appel à la procédure propagation-unitaire qui propage tous les mono-littéraux jusqu'à ce que la liste de ces derniers $Lmono$ se vide. Puis un appel est fait pour traiter les littéraux purs qui à leur tour peuvent induire des mono-littéraux. Quand il n'y a plus de mono-littéraux ou des littéraux purs à assigner, l'algorithme essaye de produire une clause $c_i \in C_{STB}$ (proposition 2). Si on arrive à produire une clause $c_i \in C_{STB}$, alors la seconde branche du point de choix courant sera explorée. Si, par contre, aucune clause n'est produite et qu'il ne reste pas de mono-littéraux ou littéraux purs à propager, alors la seconde branche de la variable point de choix devient inutile et donc coupée. L'énumération continue par le choix du prochain littéral dans STB à assigner et ce processus est réitéré jusqu'à la satisfaction de toutes les clauses ou l'affectation de tous les littéraux du STB sans apparition de la clause vide. Dans ce cas, une extension $E = HC(\pi)_I$ est obtenue. Il ne reste plus qu'à effectuer la phase de complétion, ensuite vérifier si l' extension obtenue satisfait la condition dite discriminante et celle de maximalité en variable $not\ A_i$ pour induire un modèle stable. Le pseudo-code du schéma général de la méthode est donné dans l'algorithme 1.

Algorithm 1 Schéma général de la nouvelle méthode de recherche de modèles stables

Require: La forme clausale $l(\pi)$ d'un programme logique π
Ensure: Tous les modèles stables de π

$S = \emptyset$
repeat
  while $STB \neq \emptyset$ et 'pas de conflit' **do**
    while $Lmonos \neq \emptyset$ or $Lpure \neq \emptyset$ **do**
      unit-propagation($HC(\pi)$,$Lmonos$,I) ;
      inférence($HC(\pi)$,$Lpure$,I) ;
      clause-production($HC(\pi)$) ;
    end while
    choisir littéral ;
  end while
  if pas de conflit **then**
    $E = HC(\pi)_I$ est une extension candidate ;
    E= complétion(E) ;
    if Conditions(E) **then**
      M=Atomes-positive(E) ;
      $S = S \cup M$;
    end if
  else
    backtrack
  end if
until Tout l'espace de recherche est exploré

Si n est le nombre de variables de la représentation clausale $HC(\pi)$ du programme π, k le cardinal de l'ensemble STB et m le nombre de clauses de $HC(\pi)$, alors la complexité temporelle de l'algorithme dans le pire des cas est approximativement O($knm2^k$). Nous pouvons remarquer que le facteur exponentiel de la fonction de complexité dépend du nombre k représentant la taille de l'ensemble STB et ne dépend pas du nombre de variables n comme dans les autres solveurs ASP. La valeur de k est généralement plus petite que celle de n, d'où une meilleure complexité temporelle.

Contrairement à la plupart des solveurs ASP qui utilisent la complétion Clark avec la gestion des boucles et qui donc ont une complexité spatiale exponentielle dans le pire des cas, notre méthode fonctionne à espace constant. En effet, la méthode utilise la forme clausale $HC(\pi)$ dont la taille est identique à celle du programme initial π et qui ne varie pas pendant les exécutions. La complexité spatiale est constante, elle est d'ordre $O(\mid HC(\pi) \mid) = O(\mid \pi \mid)$ dans le pire des cas.

4 Définition et propriétés de la symétrie

La notion de symétrie est très largement étudié dans le domaine de la programmation par contraintes. Nous donnerons dans ce qui suit les principales définitions et propriétés de la symétrie dans le cadre de la représentation clausale $HC(\pi)$ d'un programme logique π. Nous définissons d'abord la symétrie sémantique :

Définition 3 *Soit $HC(\pi)$ la représentation clausale de π et $L_{HC(\pi)}$ l'ensemble des littéraux de $HC(\pi)$. Une symétrie sémantique σ de $HC(\pi)$ est une permutation définie sur l'ensemble $L_{HC(\pi)}$, tel que $HC(\pi)$ et $\sigma(HC(\pi))$ ont les mêmes modèles stables.*

En d'autres termes, une symétrie sémantique de la représentation clausale d'un programme logique est une permutation de ses littéraux qui préserve les modèles stables. Nous allons maintenant définir la symétrie syntaxique :

Définition 4 *Soit $HC(\pi)$ la représentation clausale de π et $L_{HC(\pi)}$ l'ensemble de littéraux de $HC(\pi)$. Une symétrie syntaxique σ de $HC(\pi)$ est une permutation définie sur $L_{HC(\pi)}$, tel que $HC(\pi) = \sigma(HC(\pi))$.*

En d'autres mots, une symétrie syntaxique de la représentation clausale d'un programme logique est une permutation de ses littéraux qui laisse toutes les clauses du programme inchangé.

Exemple 2 *On considère le programme logique π de l'exemple 1 où $L_{HC(\pi)} = \{a, b, not\, a, not\, b\}$. La permutation $\sigma = (a, b)(nota, notb)$ définie pour $L_{HC(\pi)}$ est une symétrie syntaxique de $HC(\pi)$ ($\sigma(HC(\pi)) = HC(\pi)$).*

Définition 5 *Les deux littéraux l et l' de $L_{HC(\pi)}$ sont symétrique s'il existe une symétrie σ de $HC(\pi)$ tel que $\sigma(l) = l'$.*

Nous allons définir l'orbite d'un littéral :

Définition 6 *Soit $HC(\pi)$ la représentation clausale de π, l'orbite du littéral $l \in L_{HC(\pi)}$ sur lequel le groupe de symétrie $(Sym(HC(\pi)), \star)$ est appliqué est $l^{Sym(HC(\pi))} = \{\sigma(l) : \sigma \in Sym(HC(\pi))\}$*

Nous donnons ci-dessous une propriété qui relie entre la symétrie syntaxique et la symétrie sémantique :

Proposition 3 *Chaque symétrie syntaxique de la représentation clausale $HC(\pi)$ est une symétrie sémantique de $HC(\pi)$.*

Preuve 3 *Il est trivial de voir qu'une symétrie syntaxique de $HC(\pi)$ est toujours une symétrie sémantique de $HC(\pi)$. En effet, si σ est une symétrie syntaxique de $HC(\pi)$, alors $\sigma(HC(\pi)) = HC(\pi)$, il en résulte que $HC(\pi)$ et $\sigma(HC(\pi))$ ont les mêmes modèles stables.*

Si I est un modèle stable de $HC(\pi)$ et σ une symétrie syntaxique, nous pouvons obtenir un autre modèle stable de $HC(\pi)$ en appliquant σ sur les littéraux qui apparaissent dans I. Formellement :

Proposition 4 *Soit σ une symétrie syntaxique de $HC(\pi)$, I est un modèle stable de π ssi $\sigma(I)$ est un modèle stable de π.*

Exemple 3 *Dans l'exemple 2, l'orbite du littéral* a *est* $a^{Sym(L_\pi)} = \{a, b\}$ *et celui du littéral* $not\, a$ *est* $not\, a^{Sym(L_\pi)} = \{not\, a, not\, b\}$. *Tous les littéraux d'une même orbite sont tous symétriques. Par exemple, dans l'exemple 1 il y a deux modèles stables symétriques de* $HC(\pi)$. *Le premier c'est* $M_1 = \{b\}$ *et le second c'est* $\sigma(M_1) = \{a\}$. *Ce sont des modèles stables symétriques de* $HC(\pi)$.

Si $(Perm(HC(\pi)), \star)$ désigne le groupe de permutations de $L_{HC(\pi)}$ et $Sym(L_{HC(\pi)}) \subset Perm(L_{HC(\pi)})$ le sous-ensemble de permutations de $L_{HC(\pi)}$ formant les symétries syntaxiques de $HC(\pi)$, alors $(Sym(HC(\pi)), \star)$ est un sous-groupe de $(Perm(HC(\pi)), \star)$ représentant le groupe des symétries de $HC(\pi)$.

5 Détection des symétries

La détection des symétries est basée sur la recherche d'automorphismes de graphes. Nous commençons par représenter l'ensemble de clause de Horn $HC(\pi)$ par un graphe coloré $G_{HC(\pi)}$ puis calculer son ensemble d'automorphismes qui devrait être identique au groupe des symétries de $HC(\pi)$. Cette technique est très connue en logique propositionnelle [8, 1].

Soit $G_{HC(\pi)} = (V, E)$ le graphe associé à $HC(\pi)$, où V est un ensemble de sommets et $E \subseteq V \times V$ un ensemble d'arêtes. Un automorphisme (symétrie) de $G_{HC(\pi)}$ est une permutation des sommets qui laisse le graphe inchangé. Seul les sommets ayant la même couleur peuvent être permutés ensemble. Soit $HC(\pi)$ la représentation clausale de π, le graphe coloré associé $G_{HC(\pi)}(V, E)$ de $HC(\pi)$ est définit comme suit :

- Chaque littéral positive A_i de $HC(\pi)$ est représenté par un sommet $A_i \in V$ de couleur 1 dans $G_{HC(\pi)}$. Le littéral négatif $\neg A_i$ associé à A_i est aussi représenté par un sommet $\neg A_i$ de couleur 1 dans $G_{HC(\pi)}$. Ces deux sommets sont reliés par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque littéral $not\, A_i \in STB$ associé à A_i est représenté par un sommet $not\, A_i$ de couleur 2 dans $G_{HC(\pi)}$. Ce sommet est connecté a celui représentant A_i par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque littéral $not\, A_i \notin STB$ associé à A_i est représenté par un sommet $not\, A_i$ de couleur 3 dans $G_{HC(\pi)}$. Ce sommet est connecté a celui représentant A_i par une arête de E dans le graphe $G_{HC(\pi)}$.
- Chaque clause représentant une règle $c_i \in CR$ de $HC(\pi)$ est représenté par un sommet $c_i \in CR$ de couleur 4 dans $G_{HC(\pi)}$. Une arête connecte ce sommet c_i à ceux représentant les sommets de ses littéraux.
- Chaque clause représentant une exclusion mutuelle $c_i \in ME$ de $HC(\pi)$ est représenté par un sommet $c_i \in V$ de couleur 5 dans $G_{HC(\pi)}$. Une arête connecte ce sommet c_i aux deux sommets représentant ses littéraux.

Cette construction de graphe garantit que seuls les sommets ayant la même couleur pourraient être permutés ensemble. Le graphe $G_{HC(\pi)}$ préserve le groupe de symétries syntaxique de $HC(\pi)$. C'est à dire, le groupe de symétrie syntaxique de la représentation $HC(\pi)$ d'un programme logique π est identique au groupe d'automorphisme de son graphe $G_{HC(\pi)}$. Nous pourrions utiliser ensuite un système de détection d'automorphisme comme $saucy$, $autom$ ou $nauty$ pour détecter le groupe de symétrie syntaxique de $HC(\pi)$. Ces systèmes retournent un ensemble de générateurs du groupe de symétrie à partir desquelles nous pouvons déduire chaque symétrie de $HC(\pi)$. Notre approche est différente de celle présentée dans [11]. Cette méthode utilise la représentation body-atom d'un programme logique pour encoder un graphe orienté. Elle est également différente de la technique introduite dans [5] qui utilise un solveur ASP basé sur l'approche SAT et sur la complétion de Clark.

6 Elimination des symétries

Soit un programme logique π, les symétries de sa représentation clausale $HC(\pi)$ induisent des classes d'équivalence dans l'espace de solutions / no-goods du programme. Toutes les interprétations symétriques d'une solution / no-good I de π sont des solutions / no-goods de π. La symétrie est alors une relation d'équivalence qui partitionne l'ensemble des interprétations. Il est alors possible de représenter chaque classe d'équivalence par une interprétation représentante. Ici nous utilisons la technique du *lex leader* [2] qui ordonne tous les littéraux de $HC(\pi)$ selon un ordre lexicographique, et sélectionne la plus petite interprétation lexicographique de chaque classe d'équivalence. Soit $I = \{not\, A_1, not\, A_2, ...not\, A_n\}$ l'ensemble STB ordonné lexicographiquement. Les prédicats d'élimination des symétries (SBPs) sont construits en fonction du groupe de symétrie détecté $Sym(HC(\pi)) = \{\sigma 1, \sigma 2, ..., \sigma_n\}$ de $HC(\pi)$ avec l'ordre suivant $not\, A_1 \leq not\, A_2 \leq ... \leq not\, A_n$ sur les variables $not\, A_i$ du STB. Nous générons des SBP partielle qui élimine les symétries correspondant aux générateurs du groupe de symétrie. Les SBP partielles sont données par l'ensemble de formules suivante :

$$PC(\sigma) = \left[\bigwedge_{1 \leq k \leq m} (p_{k-1} \rightarrow (not\, A_k \leq not\, A_k^\sigma)) \right] \wedge \left[\bigwedge_{1 \leq k \leq m-1} (p_{k-1} \rightarrow (not\, A_k \geq not\, A_k^\sigma) \rightarrow p_k) \right]$$

$PC(\sigma)$ est le prédicat de permutation correspondant au générateur de symétrie σ où p_i est une variable auxiliaire définie comme suit :

$$p_i = g_{prev(i,I)} \rightarrow \bigwedge_{k \in K} \left[\bigwedge_{j \in prev(k,K)} g_j \right] \rightarrow l_k$$

Nous avons deux prédicats d'ordre $l_i = (not\, A_i \leq not\, A_i^\sigma)$ et $g_i = (not\, A_i \geq not\, A_i^\sigma)$. La fonction $g_{prev(i,I)}$ donne le prédécesseur de la variable auxiliaire p_i dans l'ensemble I. C'est à dire, $g_{prev(i,I)} = p_{i-1}$.

$$PLL(Sym(HC(\pi)) = \bigwedge_{\sigma \in Gen(Sym(HC(\pi))} PC(\sigma)$$

$PLL(Sym(HC(\pi))$ représente la formule d'élimination partielle des symétries.

7 Expérimentation

Sur la base de l'algorithme présenté précédemment, nous avons implémenté une première version d'un nouveau solveur ASP que nous désignons par $HC-asp$ pour signifier Horn Clause ASP. Nous avons enrichi ce système en lui rajoutant l'élimination des symétries pour obtenir la variante $HC-asp-sym$. La méthode d'élimination des symétries utilisée ici est une approche statique qui élimine les symétries dans une phase de prétraitement. Notre système ASP prend en entrée un programme logique terminal π produit par le grounder $gringo$ [14]. Ce programme logique est exprimé dans sa forme clausale $HC(\pi)$. Le système construit un graphe coloré $G_{HC(\pi)}$ qui représente $HC(\pi)$ que $saucy$ [9] utilise pour détecter le groupe des symétries de $HC(\pi)$. Les prédicats d'élimination partielle des symétries sont calculés en fonction du groupe des symétries détectées puis ajoutés au codage $HC(\pi)$. Après cela, un solveur ASP pourrait être utilisé en tant que boîte noire sur l'ensemble de clauses résultant.

Pour montrer l'efficacité de notre solveur $HC - asp$, nous l'avons comparé à d'autres systèmes ASP existants. Nous avons considéré dans la comparaison le solveur $Cmodels$ (version 3.86 avec $zChaff$ comme solveur SAT). Nous avons également considéré deux autres solveurs ASP connus qui sont $Smodels$ (version 2.34) et $Clasp$ (version 3.3.3). Tous les systèmes sont appliqués pour rechercher tous les modèles stables et $gringo$ est utilisé comme grounder pour chacun d'eux. Les programmes fonctionnent sur une machine Ubuntu (16,10) de 4 Go avec un processeur Intel Core i5 (1,70 GHz x 4). La durée d'exécution est limitée à 24H pour tous les systèmes expérimentés. Nous avons expérimenté différents problèmes hautement combinatoires. Pour chacun d'entre eux, nous avons progressivement augmenté la taille du problème. Le nombre de modèles stables obtenus par tous les systèmes s'ils terminent l'exécution sont donné dans la colonne "Modèles stables" des Tableaux 1,2 et 3. Nous rapportons ici les résultats obtenus sur quelques benchmarks connus qui sont le problème d'accessibilité, le problème des Pigeons, le problème de Ramsey, le problème des n-reines, le circuit hamiltonien et le problème de Schur. La tâche la plus importante dans l'ASP consiste à énumérer tous les modèles stables d'un programme logique. Nous avons choisi ces benchmarks en raison de leur nombre important de modèles stables. Ils sont très appropriés pour étudier le comportement de chacun des solveurs lorsque le nombre de modèles stables et la taille du problème augmentent. Les tableaux 1, 2 et 3 donnent le temps d'exécution des différents benchmarks pour tous les solveurs ASP. Ils donnent également des informations sur le système $HC-asp-sym$: #sym représente le nombre de symétries détectée, #SBP le nombre de SBPs ajoutés et #nssmodels le nombre de modèles sables non symétriques. Le temps d'exécution est celui de l'énumération de toutes les solutions incluant le pré-traitement sur la symétrie.

Les résultats obtenus pour la recherche de tous les modèles stables des quatre benchmarks Accessibilité(1-4), Pigeons(9-16), Ramsey(4-8) et le problème de Schur (17-28) sont dans le tableau 1. En général, $HC - asp$ surpasse tous les autres systèmes ASP. Nous pouvons observer que l'usage de l'élimination des symétries réduit considérablement l'espace des solutions de tous les benchmarks, et de ce fait, réduit le temps d'exécution. C'est à dire, $HC - asp - sym$ a de meilleurs résultats que ceux de $HC - asp$ et ceux des autres systèmes. Clairement, nous pouvons voir que l'élimination des symétries réduit significativement le nombre de modèles stables trouvés. Le gain augmente lorsque la taille du problème et le nombre de modèles stables augmentent aussi. Le problème des pigeons est une bonne illustration de cette observation. Pour le problème des pigeons de taille 9 à 11, l'espace des solutions est compressé par environ 95%. Cela réduit considérablement le temps d'exécution. Pour le problème de Ramsey et celui de Schur, nous pouvons voir que $HC - asp$, $Clasp$, et $Smodels$ ont des résultats comparable. De nouveau, $HC - asp - sym$ a de meilleurs résultats sur ces problèmes.

L'autre benchmark que nous avons expérimenté est celui des n-reines. Les résultats obtenus sont présentés dans le tableau 2. Nous pouvons voir que toutes les méthodes ont résolu efficacement les petites instances (10 à 12 reines). $HC - asp$ surpasse de manière significative toutes les autres méthodes lorsque la taille du problème augmente. L'élimination des symétries a grandement contribué à améliorer les résultats de $HC - asp$. Le nombre de solutions trouvés et le temps consacré à l'exploration de tout l'espace de recherche sont réduits quand on élimine les symétries dans $HC - asp - sym$.

Le dernier benchmark est celui du circuit hamiltonien [23]. Nous avons trouvé tous les circuits hamiltoniens de quelques graphes orientés complets dont le nombre de sommets varie de 5 à 12.Les comportements de tous les solveurs sont représentés dans le tableau 3. Les résultats

TABLE 1 – Les résultats obtenus sur les problèmes d'Accessibilité, Ramsey, Pigeons et Schur

N°	Instances	#Modèles stables					HC-asp-sym		
			HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#nssmodels	Temps
1	R_2	1	0.0005	0.0001	0.0002	0.0003	1	6	1	0.0002
2	R_3	18	0.0015	0.001	0.0005	0.015	2	23	9	0.0004
3	R_4	1606	0.030	0.070	0.022	0.091	2	54	725	0.011
4	R_5	565080	7.12	12.59	7.62	9991.28	3	134	176733	2.94
5	R_4_5_5	957	0.011	0.010	0.014	0.049	3	32	232	0.003
6	R_4_5_6	27454	0.2	0.5	0.33	18.62	5	71	6021	0.054
7	R_4_5_7	1452289	12.64	28.76	18.00	•	5	99	316803	3.34
8	R_4_5_8	137578233	1625.14	3329.66	2219.19	•	6	155	12365132	167.71
9	pi4/4	24	0.007	0.009	0.002	0.003	4	84	6	0.0005
10	pi5/5	120	0.02	0.02	0.008	0.01	5	147	12	0.0011
11	pi6/6	720	0.06	0.06	0.04	0.07	6	189	70	0.0048
12	pi7/7	5040	0.23	0.55	0.30	0.87	7	249	840	0.069
13	pi8/8	40320	1.65	4.01	2.5	52.34	8	336	2424	0.42
14	pi9/9	362880	21.63	47.11	34.60	4591.87	9	702	19634	1.26
15	pi10/10	3628800	210.14	494.40	369.80	•	10	846	170354	19.92
16	pi11/11	39916800	2728.53	6936.96	4247.58	•	11	1044	893760	55.77
17	Schur4/10	2964	0.16	0.12	0.10	0.36	8	158	325	0.0005
18	Schur4/11	8326	0.47	0.36	0.28	1.33	8	204	847	0.0064
19	Schur4/12	18539	0.98	0.82	0.65	4.82	9	272	1687	0.025
20	Schur4/13	48987	2.6	3.26	1.77	39.04	9	350	2358	0.5
21	Schur4/14	95311	5.29	5.7	4.43	169.85	10	402	5863	1.3
22	Schur4/15	247147	14.20	15.61	12.36	1286.12	10	498	10354	3.68
23	Schur4/16	408642	26.22	27.58	24.10	3500.16	11	535	20789	6.8
24	Schur4/17	920149	65.61	66.84	49.84	15976.8	11	602	25893	10.23
25	Schur4/18	1597576	134.87	137.89	102.88	48134.4	12	647	45283	20.34
26	Schur4/19	3344347	287.85	301.96	207.86	•	12	732	60352	36.22
27	Schur4/20	3832609	406.88	436.88	292.91	•	13	802	65821	80.34
28	Schur4/21	7924530	965.80	987.62	707.90	•	13	897	92358	120.85

TABLE 2 – Les résultats obtenus sur le problèmes des n-reines

#Taille	#Modèles stables					HC-asp-sym			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Temps
10	724	0.68	0.23	0.66	0.27	3	468	240	0.10
11	2680	2.79	1.02	2.95	2.48	3	528	852	0.56
12	14200	12.2	8.75	15.19	41.44	3	810	4981	3.69
13	73712	79.55	122.91	87.69	1642.93	3	921	24934	20.53
14	365596	371.73	2631.83	496.98	•	3	918	107371	120.26
15	2279184	2797.02	34337.21	3352.37	•	3	1221	788141	854.31
16	14772512	12087.40	•	23134.22	•	3	1188	4310853	4976.44
17	95815104	87088.00	•	•	•	3	1275	29227320	29429.41

TABLE 3 – Les résultats obtenus sur le problème du circuit hamiltonien

#Taille	#Modèles stables					HC-asp-sym			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Temps
5	24	0.012	0.003	0.003	0.003	3	90	6	0.0005
6	120	0.026	0.016	0.014	0.02	3	132	48	0.0031
7	720	0.09	0.10	0.08	0.12	4	186	288	0.014
8	5040	0.64	0.78	0.67	1.69	4	246	1992	0.13
9	40320	5.76	7.87	6.00	80.75	5	348	5040	0.64
10	362880	66.67	89.94	72.25	6177.89	5	396	40320	6.63
11	3628800	910	1141.84	964.15	•	6	591	412118	107.26
12	39916800	13173.42	22263.37	15964.15	•	6	612	7327392	4604.26

montrent que toutes les méthodes ont résolu de manière efficace les instances ayant un nombre de sommets inférieur à huit (petites instances). $HC-asp-sym$ obtient de meilleurs résultats que les autres méthodes. Le bénéfice de la symétrie est plus important lorsque la taille du problème augmente.

8 Conclusion

Dans cet article, nous avons fourni une nouvelle méthode pour rechercher des modèles stables basée sur une sémantique relativement nouvelle introduite dans [6]. Cette méthode a l'avantage d'utiliser une représentation en clause de Horn dont la taille est identique à celle du programme logique source. Elle a une complexité spatiale constante. La sémantique utilisée prévient la méthode de la lourdeur induite par la gestion des boucles faite par tous les solveurs basé sur la complétion de Clark. L'autre avantage de notre approche est le processus énumératif simplifié qui est effectué uniquement sur un sous-ensemble de littéraux représentant le strong backdoor du programme logique. Cela conduit à une réduction considérable de la complexité temporelle. Nous avons également proposé l'intégration

de l'élimination des symétries afin d'éviter d'explorer des sous-espaces isomorphes. Nous avons expérimenté la méthode proposée avec et sans élimination des symétries sur une variété de problèmes combinatoires connus et les résultats obtenus ont montré que notre approche est une bonne alternative pour implémenter des solveurs ASP et que l'élimination des symétries conduit à une amélioration significative de cette méthode.

Nous envisageons par la suite d'intégrer l'élimination des symétries de manière dynamique et comparer par rapport à l'approche statique. L'extension de notre approche à d'autres classes de programmation logique ou à des parties de certaines logiques non monotones plus générales est aussi envisageable.

Références

[1] F.A Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Solving difficult sat instances in the presence of symmetry. *DAC*, pages 731–736, 2002.

[2] F.A Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Solving difficult sat instances in the presence of symmetry. *DAC*, pages 1117–1137, 2003.

[3] B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application. *CADE*, 607 :281–294, 1992.

[4] B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus. *The Journal of Automated Reasoning*, pages 89–102, 1994.

[5] Belaid Benhamou. Dynamic and static symmetry breaking in answer set programming. *LPAR*, pages 112–126, 2013.

[6] Belaïd Benhamou and Pierre Siegel. A new semantics for logic programs capturing and extending the stable model semantics. *Tools with Artificial Intelligence (ICTAI)*, pages 25–32, 2012.

[7] Keith L Clark. Negation as failure. *Logic and data bases*, pages 293–322, 1978.

[8] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. *KR*, pages 148–159, 1996.

[9] P.T Darga, K.A. Sakallah, and I.L. Markov. Faster symmetry discovery using sparsity of symmetries. *DAC*, pages 149–154, 2008.

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

[11] Christian Drescher, Oana Tifrea, and Toby Walsh. Symmetry-breaking answer set solving. *AI Communications*, 24 :177–194, 2011.

[12] Francois Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1 :51–60, 1994.

[13] Martin Gebser, Benjamin Kaufmann, Andrá Neumann, and Torsten Schaub. Conflict-driven answer set solving. *IJCAI*, 7 :386–392, 2007.

[14] Martin. Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. *International Conference on Logic Programming and Nonmonotonic Reasoning*, 7 :266–271, 2007.

[15] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. *ICLP/SLP*, 50 :1070–1080, 1988.

[16] Tarek Khaled, Belaid Benhamou, and Pierre Siegel. Vers une nouvelle méthode de calcul de modéles stables et extensions en programmation logique. *JIAF*, pages 151–160, 2017.

[17] B. Krishnamurty. Short proofs for tricky formulas. *Acta Inf.*, 22 :253–275, 1985.

[18] Nicola. Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7 :499–562, 2006.

[19] Yuliya Lierler and Marco Maratea. Cmodels-2 : Sat-based answer set solver enhanced to non-tight programs. *Logic Programming and Nonmonotonic Reasoning*, pages 346–350, 2004.

[20] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas ? *ACM Transactions on Computational Logic (TOCL)*, 7 :261–268, 2006.

[21] Fangzhen Lin and Yuting Zhao. Assat : Computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, pages 115–137, 2004.

[22] B. McKay. Practical graph isomorphism. *Numerical mathematics and computing.*, pages 45–87, 1981.

[23] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25 :241–273, 1991.

[24] J.F. Puget. Automatic detection of variable and value symmetries. *CP*, pages 475–489, 2005.

[25] Patrik Simons, Ilkka Nimelä, and Timo Soininen. Extending and implementing the stable model semantic. *Artificial Intelligence*, 138 :181–234, 2002.

[26] Ryan Williams, Carla P Gomes, and Bart Selman. Backdoors to typical case complexity. *International joint conference on artificial intelligence*, 18 :1173–1178, 2003.