# Vers une nouvelle méthode de calcul de modèles stables et extensions en programmation logique

**Tarek Khaled**[1] **Belaïd Benhamou**[1] **Pierre Siègel**[2]

[1] Université Aix Marseille, LSIS UMR 7296, Marseille, France
[2] Université Aix-Marseille, LIF UMR 7279, Marseille, France

{tarek.khaled,belaid.benhamou}@lsis.org pierre.siegel@lif.univ-mrs.fr

**Résumé**

La programmation par ensembles réponses (Answser set programmning (ASP)) est une approche très connue en programmation déclarative. Elle est très utilisée dans le domaine de la représentation des connaissances et le raisonnement. L'ASP est un sujet de recherche très prisé ces dernières années. Les solveurs ASP sont devenus très performants et compétitifs, ils ont gagné leurs galons dans les applications industrielles. Le travail que nous présentons dans cet article porte sur l'élaboration d'un algorithme de recherche d'ensembles réponses basé sur une nouvelle sémantique (Benhamou et al. 2012) des programmes logiques qui capture et qui étend celle des modèles stables (Gelfond et al. 1988). Cet algorithme est basé sur une énumération au sens de la procédure DPLL adaptée au cadre de l'ASP et à la nouvelle sémantique utilisée. L'avantage de cette méthode est qu'elle opère sur un système de clauses de Horn ayant la même taille que le programme logique source et à espace constant. Elle évite ainsi la lourdeur de la complétion de Clark et la gestion des boucles souvent utilisées dans les solveurs ASP basés sur DPLL. De plus, l'énumération est faite sur une restriction de l'ensemble des variables représentant le strong backdoor (STB) du programme logique considéré. La complexité de l'algorithme est calculée en fonction de cet ensemble (STB) et son efficacité en dépend. Nous avons introduit quelques nouvelles règles d'inférence dans ce cadre que l'algorithme utilise pour faire des coupures dans l'arbre de recherche et réduire ainsi sa taille. Cette méthode permet en plus des modèles stables, de générer des extra-modèles exprimant l'extension apportée à la sémantique de Gelfond et al. Nous avons implémenté une première version de cette méthode qui permet de calculer les modèles stables d'un programme logique que nous avons testée dans un premier temps sur des instances de coloriage de graphes aléatoires.

**Abstract**

Answers set programming (ASP) is a well-known approach in declarative programming. It is widely used in the field of knowledge representation and reasoning. ASP is a highly sought research topic in recent years. ASP solvers have become very efficient and competitive, they have gained their stripes in industrial applications. The work we present in this paper focuses on the development of an algorithm for finding answer sets that is based on a new semantics (Benhamou et al., 2012) of logical programs that captures and extends that one of stable models (Gelfond 1988). This algorithm is based on an enumeration in the sense of the DPLL procedure adapted to the framework of the ASP and to the new semantics used. The advantage of this method is that it operates on a Horn clause system having the same size as the source logic program. It thus, avoids the heaviness of the Clark completion and the loop management often used in ASP solvers based on DPLL. Moreover, the enumeration is done on a restriction of the set of variables of the set of clauses representing the strong backdoor (STB) of the considered logic program. The complexity of the algorithm is calculated as a function of this set (STB) on which its efficiency depends. We have introduced some new inference rules in this framework that the algorithm uses to make cuts in the search tree and thus reduce its size. This method also allows in addition to the stable models, to generate extra-models expressing the extension given to the semantics of Gelfond and al. We have implemented a first version of this method which allows to calculate the stable models of a logic program that we tested on random graph coloring instances.

## 1 Introduction

Un programme logique $\pi$ est un ensemble de faits et de règles se terminant par un point ('.'). Les connecteurs ' :-' et ',' apparaissent dans ces règles et peuvent être interprétés comme étant 'si' et 'et' respectivement. Précisément, le

programme logique $\pi$ est un ensemble de règles de la forme *r :tête(r)*←*corps(r)*, où $corps(r)$ représente l'ensemble des prémisses de la règle *r* et *tête(r)* sa partie conclusion ou conséquence. Cet ensemble de prémisses est en général une conjonction de littéraux où il peut y avoir des négations classiques et des négations par échec. Cette conjonction est donnée sous forme de liste de littéraux séparée par des virgules. La partie *tête(r)* est exprimée soit par un seul littéral, ou par une disjonction de littéraux. La lecture intuitive de ces règles nous amène à dire que la partie *tête(r)* peut être inférée si la partie $corps(r)$ a été déjà prouvée.

La Programmation par ensembles réponses (ASP) [17] est une forme de programmation déclarative non monotone qui offre un cadre idéal pour la formulation de problèmes en Intelligence Artificielle. Cela peut être du raisonnement de sens commun, du web sémantique, du raisonnement causal, ou de la résolution de problèmes combinatoires tels que la planification, des problèmes de théorie des graphes, la configuration, ou les problèmes de bio-informatique [5]. Le paradigme de programmation ASP fournit un cadre général pour la résolution de problème de décision et d'optimisation [16]. L'approche a été rendue populaire grâce au développement de plusieurs solveurs performants dédiés à l'ASP comme *smodels* [13], *DLV* [11] et dernièrement *Clasp* [15] et notamment ceux basés sur la procédure DPLL et les solveurs SAT comme $ASSAT$ [7].

L'idée de base de l'ASP est d'exprimer un problème donné sous forme d'un programme logique pour lequel on doit chercher les modèles stables qui représentent les solutions du problème originel [10]. Pour avoir une expression concise du problème, on utilise dans un premier temps, des règles construites à base de la logique du premier ordre. En conséquence, le problème sera exprimé par un programme logique dit souvent programme non terminal contenant des prédicats avec des variables. Des systèmes appelés en anglais "grounders" ont été conçus pour calculer l'ensemble des instances terminales des règles de ce programme non terminal donnant en conséquence un programme logique terminal équivalent. La figure 1 illustre le fonctionnement général d'un système ASP.

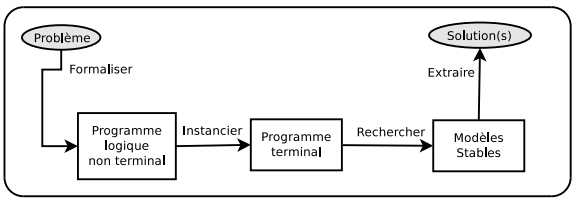


FIGURE 1 – Modèle conceptuel du paradigme ASP

La sémantique des modèles stables [8] est l'une des plus utilisée dans le cadre ASP. Il existe dans la littérature plusieurs sémantiques pour les programmes logiques. La première sémantique introduite dans ce cadre, est la complétion de Clark [4]. Puis, d'autres sémantiques telles que celle de "well-founded" [1], celle des modèles stables [8] ou celle d'ensembles réponses [12] ont été introduites. Ces sémantiques ont toutes pour but de donner une signification aux programmes logiques considérés. Notamment, elles essayent de donner un sens à la négation par échec présente dans les règles du programme. C'est souvent ce sens qui les définit et qui les distingue.

Dans le cadre de ce travail nous utilisons la nouvelle sémantique introduite dans [2]. Nous avons choisi cette sémantique pour sa simplicité et les avantages qu'elle offre. Cette sémantique est basée sur une notion d'extension qu'elle définit pour les programmes logiques. Dans cette sémantique, les programmes sont représentés par un ensemble de clauses de Horn ayant la même taille que le programme d'entrée. Elle a l'avantage de caractériser les modèles stables en utilisant cette représentation Horn clausale. Cette représentation clausale permet d'avoir des algorithmes de résolution avec de bonnes propriétés de complexité. L'algorithme que nous allons présenter utilise cette représentation et tire profit de tous les avantages qu'offre un ensemble de clauses de Horn. Elle évite ainsi la lourdeur de la complétion de Clark [4] souvent utilisées dans les solveurs ASP basés sur DPLL et la gestion des boucles qui les rend non constant en complexité spatiale. C'est une adaptation de la procédure DPLL au cadre de l'ASP sous la nouvelle sémantique. Cet algorithme est basé sur une énumération faite sur une restriction de l'ensemble des variables représentant le strong backdoor [1] (STB) [14] du programme logique considéré. La complexité de l'algorithme est calculée en fonction de cet ensemble (STB) et son efficacité en dépend. Nous utilisons quelques nouvelles règles d'inférence que l'algorithme exploite pour élaguer l'arbre de recherche. Cette méthode dans sa globalité, permet de calculer des extensions, à partir desquelles on peut générer des modèles stables ou des extra-modèles exprimant ainsi l'extension apportée à la sémantique de Gelfond et al. [8].

Le reste de l'article est organisé comme suit : dans la section 2, nous rappelons quelques notions de base sur la programmation ASP et les fondements de la nouvelle sémantique [2] que nous utilisons dans la nouvelle méthode de résolution. Nous présenterons cette méthode dans la section 3. Dans la section 4, nous donnons les premiers résultats expérimentaux de notre approche et enfin nous concluons le travail en section 5.

## 2 Préliminaires

Un programme logique $\pi$ est composé par un ensemble de règles de la forme *r :tête(r)* ← *corps(r)*. Il existe différentes classes de programmes logiques. Ils se distingues par la présence ou l'absence de la négation classique et

1. Intuitivement, un ensemble de variables d'un problème forme un strong backdoor si le problème restant après l'instanciation de ces variables est résolu en temps polynomial

de la négation par échec dans l'ensemble des règles qui les composent. Un programme logique positif $\pi$ est un ensemble de règles de la forme : $r = A_0 \leftarrow A_1, A_2, ..., A_m$, avec $(m \geq 0)$ et où $A_{i\in\{0,...,m\}}$ est un atome. Un programme positif ne contient pas de négation par échec ou de négation classique. Il est connu que tout programme positif admet un seul modèle stable. Il est le plus petit ensemble d'atomes obtenu par l'application des règles du programme jusqu'à un point fixe. C'est le modèle minimal de Herbrand de l'ensemble de clauses de Horn formant le programme $\pi$ ; il est noté $Cn(\pi)$.

Dans la suite de cet article, nous ne nous intéresserons qu'aux programmes logiques généraux (normaux). Un programme logique général $\pi$ 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 $A_{i\in\{0...m\}}$ est un atome et *not* le symbole exprimant la négation par échec. Le corps positif de $r$ est dénoté par $corps^+(r) = \{A_1, A_2, ..., A_m\}$ et le corps négatif par $corps^-(r) = \{A_{m+1}, ..., A_n\}$. Intuitivement, la règle $r$ est interprétée comme suit : si on prouve tous les atomes du corps positif $corps^+(r) = \{A_1, A_2, ..., A_m\}$ de la règle $r$, et qu'on n'arrive à prouver aucun des atomes du corps négatif $corps^-(r) = \{A_{m+1}, ..., A_n\}$, alors on infère $A_0$. Le réduit d'un programme $\pi$ par rapport à un ensemble d'atomes $X$ est le programme positif $\pi^X$ obtenu à partir de ground($\pi$) en supprimant :

— chaque instance de règle ayant 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 instances des règles restantes.

La sémantique la plus utilisée pour les programmes logiques généraux est celle des modèles stables [8]. Un ensemble d'atomes $X$ est un modèle stable de $\pi$ si et seulement si $X = Cn(\pi^X)$. En d'autres mots, l'ensemble d'atomes $X$ est un modèle stable de programme $\pi$ ssi $X$ est identique au modèle minimal de Herbrand du réduit $\pi^X$ du programme $\pi$ par rapport à $X$.

On trouve d'autres sémantiques pour les programmes logiques. Pour gérer la notion de "négation par échec", Clark a proposé le concept de complétion d'un programme logique (notation $comp(\pi)$). Il est connu que tout modèle stable de $\pi$ est un modèle de sa complétion mais la réciproque ne s'applique que si le programme est sans boucle (tight en anglais) [6]. Afin d'établir l'équivalence entre la sémantique d'un programme et sa complétion, des formules de traitement des boucles sont ajoutées à la complétion (Fangzhen Lin et Yuting Zhao [7]). Le nombre de boucles dans le pire des cas peut varier d'une façon exponentielle, ce qui rend leur traitement infaisable[18].

Dans le cas de notre étude, nous utilisons la sémantique introduite dans [2] qui offre plusieurs avantages. Elle est basée sur une notion d'extension d'un ensemble de clauses représentant le programme d'entrée. Cet ensemble a l'avantage d'avoir la même taille que le programme logique qu'il représente. C'est une très bonne alternative à la complétion de Clark [4] utilisée par la majorité des solveur basés sur l'énumération à la DPLL. La méthode que nous présentons ici est dispensée de la lourdeur qu'implique l'utilisation de la complétion. A sa place, elle manipule un ensemble de clauses de Horn sur lequel elle effectue une énumération pour chercher les modèles stables. Pour un programme logique donné, la méthode calcule une extension en ajoutant à l'ensemble de clauses de Horn qui le représentent un ensemble maximal consistant de littéraux ($not\, A_i \in STB$) de l'ensemble *strong backdoor* $STB$. Chaque modèle stable correspond à une extension du programme considéré ; la caractérisation de ce modèle se fait grâce à une condition discriminante facilement vérifiable [2].

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 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 $\pi$ 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 $\pi$ 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 :

$$\mathrm{L}(\pi) = \{\bigcup_{\mathrm{r}\in\pi}(\mathrm{A}_0 \vee \neg\mathrm{A}_1 \vee, ..., \neg\mathrm{A}_\mathrm{m} \vee \neg\mathrm{not\ A}_{\mathrm{m}+1}, ..., \neg\mathrm{not\ A}_\mathrm{n}) \bigcup_{\mathrm{A_i}\in\mathrm{V}} (\neg\mathrm{A_i} \vee \neg\mathrm{not\ A_i})\}.$$

Nous pouvons remarquer que la taille du codage $L(\pi)$ est de l'ordre de $taille(\pi) + 2n$, $n$ étant le nombre de variables propositionnelles de $\pi$ appartenant à $V$. Le facteur $2n$ dans la taille de $L(\pi)$ correspond à l'ensemble de clauses $ME$ représentant l'axiome d'exclusion mutuelle. Mais ce dernier peut être implémenté comme une règle d'inférence sans avoir besoin de mémoriser l'ensemble de clauses $ME$.

La méthode que nous allons décrire travaillera alors avec la forme Horn clausale $L(\pi)$ ayant exactement la même taille que le programme $\pi$. Elle énumérera sur un sous-ensemble de variables jouant le rôle d'un strong backdoor (STB). Ce sous-ensemble est défini par $STB = \{not\, A_i : \exists r \in \pi, not\, A_i \in corps^-(r)\} \subseteq nV$. L'ensemble STB est le sous-ensemble des littéraux positifs de type $not\, A_i$ qui apparaissent dans $\pi$.

Étant donné un programme $\pi$ et son ensemble STB. Une extension de $L(\pi)$ par rapport à STB (une extension de $(L(\pi), STB)$ est l'ensemble de clauses consistant obtenu à partir de $L(\pi)$ en ajoutant un ensemble maximal de littéraux $not\, A_i \in STB$. Formellement :

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

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

**Exemple 1** *On considère le programme logique :*

$$\pi = \left\{ \begin{array}{l} a \leftarrow c, not\, b \\ b \leftarrow a \\ c \leftarrow not\, d \\ a \leftarrow \end{array} \right\}$$

*sa représentation Horn clausale est $L(\pi) = CR \cup ME$ où :*

$$CR = \{a \vee \neg c \vee \neg not\, b, b \vee \neg a, c \vee \neg not\, d, a\}$$
$$ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b, \neg c \vee \neg not\, c, \neg d \vee \neg not\, d\}$$

*Son ensemble strong backdoor est $STB = \{not\, b, not\, d\}$ et $(L(\pi), STB)$ a une seule extension $E = L(\pi) \cup \{not\, d\}$. E est maximalement consistante sur l'ensemble STB. En effet, si on rajoute not b à l'extension E, l'ensemble devient inconsistant.*

Il a été démontré dans [2] que chaque modèle stable d'un programme logique $\pi$ est représenté par une extension $E$ de sa forme logique $L(\pi)$ qui vérifie la condition discriminante $(\forall \mathrm{A_i} \in \mathrm{V}, \mathrm{E} \models \neg \mathrm{not\ A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i})$ . Certaines extensions de $L(\pi)$ ne correspondent à aucun modèle stable. Ces extra-extensions coïncident avec des extra-modèles qui représentent une extension à la sémantique des modèles stables [8]. La caractérisation d'un modèle stable se fait de manière très simple grâce à une condition simple à vérifier que doivent satisfaire les extensions correspondant aux modèles stables. Cette condition est appelée condition discriminante dans [2]. Formellement, nous avons :

**Théorème 1** *Si X est un modèle stable d'un programme logique $\pi$, alors il existe une extension E de $(L(\pi), STB)$ 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 : $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$.*

**Preuve 1** *La preuve est donnée dans [2].*

**Théorème 2** *Si $\mathrm{E}$ est une extension de $(L(\pi), STB)$, qui vérifie la condition discriminante $(\forall \mathrm{A_i} \in \mathrm{V}, \mathrm{E} \models \neg \mathrm{not\ A_i} \Rightarrow \mathrm{E} \models \mathrm{A_i})$, alors $\mathrm{X} = \{\mathrm{A_i} : \mathrm{E} \models \mathrm{A_i}\}$ est un modèle stable de $\pi$.*

**Preuve 2** *La preuve est donnée dans [2].*

**Remarque 1** *Comme toute extension E est formée par un ensemble de clauses de Horn, alors la résolution unitaire est suffisante pour déduire tout atome $A_i$ à partir de E ($E \models A_i$).*

**Exemple 2** *Soit le programme logique $\pi$ composé de deux règles :*

$$\pi = \left\{ \begin{array}{l} a \leftarrow not\, b \\ b \leftarrow not\, a \end{array} \right\}$$

*Sa représentation clausale est : $L(\pi) = CR \cup ME$ où $CR = \{a \vee \neg not\, b, b \vee \neg not\, a\}$ et où $ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b\}$. Son ensemble strong backdoor est $STB = \{not\, a, not\, b\}$. La paire $(L(\pi), STB)$ a deux extensions $E_1 = L(\pi) \cup \{not\, a\}$ et $E_2 = L(\pi) \cup \{not\, b\}$ qui vérifient la condition discriminante. On peut facilement voir que $E_1 \models \{b, \neg a, \neg not\, b\}$ et que $E_2 \models \{a, \neg b, \neg not\, a\}$. A partir de $E_1$ resp. $E_2$, on obtient, par résolution unitaire, les ensembles de littéraux positifs impliqués $X_1 = \{b\}$ resp. $X_1 = \{a\}$ qui sont les deux modèles stables du programme $\pi$.*

Comme $\pi$ est un programme logique général, toute extension de $(L(\pi), STB)$ est un ensemble consistant de clause de Horn. Donc, tous les littéraux positifs $(A_i)$ seront inférés par résolution unitaire. Ce qui réduit la complexité de l'algorithme de calcul de modèles stables résultant de la nouvelle sémantique.

## 3 Présentation de la nouvelle méthode

Nous présenterons ici un algorithme de recherche de modèles stables basé sur la nouvelle sémantique [2]. Etant donné un programme logique $\pi$, cet algorithme calcule toutes les extensions de $(L(\pi), STB)$ à partir desquelles les modèles stables seraient déduits par résolution unitaire. Intuitivement, la recherche d'extension de $(L(\pi), STB)$ se fait par l'ajout progressive de de littéraux $not\, A_i$ du STB en vérifiant à chaque ajout la consistance de l'ensemble obtenu. Ensuite, si on ne s'intéresse qu'aux modèles stables, il suffit de retenir que les extensions qui vérifient la condition discriminante. En d'autres mots, on élimine les extra-extensions qui ne vérifient pas celle-ci.

Il existe deux grandes approches dans la conception de systèmes ASP pour le calcul de modèles stables. L'une d'entre elles s'appuie sur les propriétés de la sémantique

utilisée pour concevoir le système, et l'autre transforme le programmes logique considéré en un problème de satisfaisabilité booléenne (SAT) selon la complétion de Clark pour lequel on pourrait utiliser des solveurs SAT dans la résolution. Notre algorithme fait partie de la première approche. Nous avons adapté la procédure DPLL au cadre de l'ASP traité dans la nouvelle sémantique. Le principe de notre méthode de calcul est le suivant : on construit incrémentalement une extension en alternant dans l'arbre de recherche des nœud déterministes correspondant aux propagations unitaires et des nœuds non déterministes appelés points de choix définis par l'affectation d'une valeur de vérité (vrai ou faux) à un littéral de l'ensemble strong backdoor $STB$. Nous introduisons quelques nouvelles règles d'inférence qui permettent d'augmenter le nombre de propagations unitaires et, par conséquent, réduire l'espace de recherche. L'algorithme que nous présentons permet de calculer tous les modèles stables d'un programme logique donné.

### 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.

**Definition 2** *Soit un programme $\pi$ et $L(\pi) = CR \cup ME$ sa forme clausale. On définit sur $L(\pi)$ les deux règles d'inférence suivantes :* $\frac{A_i}{\neg not\, A_i}$ *et* $\frac{not\, A_i}{\neg A_i}$.

Tout programme logique général $\pi = \{r : A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n\}, (0 \leq m < n)$ est exprimé par l'ensemble de clauses propositionnelles de Horn $L(\pi) = \{\bigcup_{r\in\pi}(r : A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_i m+1, ..., \neg not\, A_n) \bigcup_{A_i\in V} (\neg A_i \vee \neg not\, A_i)\}$ qui représente son codage $CNF$ dans le langage propositionnel $L$. Les deux règles d'inférence se justifient par la présence de l'ensemble de clauses $ME = \bigcup_{A_i\in V} \{(\neg A_i \vee \neg not\, A_i)\}$ exprimant l'exclusion mutuelle entre toute paire d'atomes $A_i$ et $\neg not\, A_i$.

Avec la considération de ces deux règles dans le raisonnement de la méthode, on pourrait supprimer de $L(\pi)$ le sous-ensemble de clauses $ME$ des exclusions mutuelles. On obtient ainsi une taille de $L(\pi)$ identique à celle du programme $\pi$. La forme clausale du programme logique $\pi$ serait réduite à : $L(\pi) = \{\bigcup_{r\in\pi}(r : A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_{m+1}, ..., \neg not\, A_n)\}$

L'utilisation et l'implémentation des deux règles d'inférence introduites précédemment dans le processus de résolution permettraient d'accroître le nombre de propagations unitaires et élagueraient l'arbre de recherche.

La complexité d'une méthode d'énumération dans un espace de recherche représenté par un arbre binaire est souvent calculée en fonction du nombre de points de choix effectués. Dans le cas de notre méthode, l'énumération est faite sur le sous-ensemble de variables $STB = \{not\, A_i : \exists r \in \pi, A_i \in r^-\}$ qui représente le strong backdoor. Soit $C_{STB} = \{c_i = \neg not\, A_{i_1} \vee, ..., \vee \neg not\, A_{i_k} / \mid c_i \mid \geq 1, \forall j \in \{1..k\}, not\, A_{i_j} \in STB\}$ l'ensemble de clauses négatives possibles formées par les littéraux du STB ayant au moins un littéral. L'assignation non déterministe d'un point de choix correspondant à la variable $not\, A_j$ est faite en lui affectant en premier lieu la valeur de vérité vrai pour favoriser la maximalité de l'extension en cours. L'exploration de la branche correspondant à l'assignation de la valeur de vérité $faux$ à $not\, A_j$ (où en affectant $\neg not\, A_j$ à vrai) n'est nécessaire que dans le cas où la première branche aurait produit au moins une sous clause $c_i \in C_{STB}$. Cette propriété, que nous allons démontrer par la suite, permettrait de réduire considérablement la complexité de l'algorithme étudié. L'algorithme vérifie a chaque affectation d'un nouveau littéral la consistance du système de clauses présent en ce nœud. Comme nous avons un ensemble de clauses de Horn, une propagation par résolution unitaire assure ce teste de consistance.

**Proposition 1** *Soit $\pi$ un programme logique, $L(\pi)$ sa forme Horn clausale, $L(\pi)_I$ 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 $\forall c_i \in C_{STB}$, $L(\pi)_I \wedge not\, A_j \not\models c_i$, alors toute extension de $L(\pi)_I \wedge \neg not\, A_j$ est aussi une extension de $L(\pi)_I \wedge not\, A_j$.*

**Preuve 3** *Le sous-ensemble de clauses $L(\pi)_I$ est le système de clauses simplifié, obtenu à partir de $L(\pi)$ par la considération des littéraux interprétés dans l'instanciation partielle I. L'ensemble de clauses $L(\pi)_I$ 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 $L(\pi)_I$ 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 $not\, A_j$. Soit $e = not\, A_{i_1} \wedge ... \wedge not\, A_{i_k}$, avec $not\, A_{i_j} \in STB$ une extension de $L(\pi)_I \wedge \neg not\, A_j$, montrons que e est aussi une extension de $L(\pi)_I \wedge not\, A_j$. On peut remarquer que $L(\pi)_I \wedge \neg not\, A_j \equiv C_2$ et que $L(\pi)_I \wedge not\, A_j \equiv C_1 \wedge C_2$. L'ensemble e est une extension de $L(\pi)_I \wedge \neg not\, A_j$, donc $C_2 \wedge e$ est consistant. Pour montrer que e est aussi une extension de $L(\pi)_I \wedge not\, A_j$, 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* $L(\pi)_I \wedge not\, A_j \models \neg not\, A_{i_1} \vee ... \vee \neg not\, Ai_k \in C_{STB}$. *Donc* $L(\pi)_I \wedge not\, A_j \models c_i \in C_{STB}$ *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. On a introduit ainsi une nouvelle coupure dans l'arbre de recherche qui pourrait réduire la complexité de l'algorithme et qui pourrait augmenter son efficacité dans la pratique.

Nous allons maintenant montrer comment exploiter l'apparition de certains littéraux purs (monotones) dans la forme clausale $L(\pi)$ du programme logique $\pi$. Nous rappelons qu'un littéral pur (monotone) est un littéral qui apparaît dans une seule forme, soit dans sa parité positive ou dans sa parité négative. Ces littéraux sont souvent ignorés dans les implémentations des solveurs SAT basé sur DPLL, mais, pour les solveur ASP, ils jouent un rôle très important.

**Proposition 2** *Soit un programme logique* $\pi$, $L(\pi)$ *sa forme clausale et* $\neg A_i$ *un littéral pur de* $L(\pi)$, *si X est un modèle stable ou un extra-modèle de* $\pi$ *alors* $\neg A_i \in X$.

**Preuve 4** *Le littéral* $\neg A_i$ *est pur dans* $L(\pi)$, *ce qui implique que* $A_i$ *n'a pas d'occurrences dans* $L(\pi)$, *donc le littéral* $A_i$ *ne pourra jamais être inféré. Par conséquent,* $A_i$ *ne pourra faire partie d'un modèle stable / extra-modèle X. Par l'application de l'hypothèse du monde clos, nous avons* $\neg A_i \in X$.

Cette proposition permet de traiter les littéraux purs comme des mono-littéraux à propager. La propagation de ces derniers contribuent à la réduction des points de choix de l'arbre et, en conséquence, son élagation.

La propriété qui va suivre est restreinte au cas de modèles stables. Elle n'est pas vérifiée dans le cas d'extra-modèles.

**Proposition 3** *Soit un programme logique* $\pi$ *et* $L(\pi)$ *sa forme clausale, si* $\neg A_i$ *est vrai dans un modèle stable X de* $\pi$ *alors not* $A_i$ *doit être vrai dans X.*

**Preuve 5** *Si* $\neg A_i$ *est vrai dans le modèle stable X, alors le seul cas où* $\neg not\, A_i$ *pourrait être produit est l'existence d'une sous-clause* $A_i \vee \neg not\, A_i$ *de* $L(\pi)_X$. *Mais dans ce cas, l'extension correspondant à X ne vérifie pas la condition discriminante. En conséquence, X ne serait n'est pas un modèle stable, ce qui est contradictoire avec l'hypothèse.*

La dernière proposition peut se traduire dans l'algorithme par la règle d'inférence ($\frac{\neg A_i}{not\, A_i}$) qui permet d'élaguer l'arbre de recherche des modèles stables. cette règle garantit la complétude de l'algorithme sur les modèles stables mais pas sur les extra-modèles. En effet, l'application de cette dernière pourrait supprimer des extra-modèles correspondant à des extra-extensions. Nous ne l'appliquons que pour la recherche de modèles stables. Toutes les propositions énoncées précédemment s'expriment par des règles d'inférence que nous avons implémentées dans la nouvelle méthode. Ces différentes règles induisent des coupures dans l'arbre de recherche et réduisent de manière considérable l'espace de recherche.

**Proposition 4** *Si* $L(\pi)$ *est la forme clausale d'un programme logique* $\pi$ *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* $L(\pi)_I$.

**Preuve 6** *D'après le théorème de déduction automatique, montrer que* $L(\pi)_I \models c_i$ *est équivalent à* $L(\pi)_I \wedge \neg c_i \models \perp$. *Comme* $L(\pi)$ *est à la base un ensemble de clauses de Horn, il en résulte que l'ensemble de clauses simplifié* $L(\pi)_I \wedge \neg c_i$ *l'est aussi car* $L(\pi)_I \wedge not\, A_{i_1} \wedge ... \wedge not\, Ai_k$ *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* $L(\pi)_I \wedge \neg c_i$. *En d'autres mots, la résolution unitaire est suffisante pour montrer* $L(\pi)_I \wedge \neg c_i \models \perp$ *et, par conséquent, suffisante pour montrer* $L(\pi)_I \models c_i$.

Pour appliquer la coupure induite par la proposition 2 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 4).

### 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 [2]. Nous avons aussi introduit et implémenté plusieurs nouvelles règles d'inférences pour booster la méthode. Dans ce nouvel algorithme, la recherche de modèle stable 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'agorithme 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 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.

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

**Entrées** La forme clausale $l(\pi)$ d'un programme logique $\pi$
**Sorties** Tous les modèles stables de $\pi$

1: **répéter**
2: **tant que** $STB \neq \emptyset$ et 'pas de conflit' **Faire**
3: **tant que** $Lmonos \neq \emptyset$ ou $Lpurs \neq \emptyset$ **Faire**
4: propagation-unitaire($L(\pi)$,$Lmonos$,$I$) ;
5: inférence($L(\pi)$,$Lpurs$,$I$) ;
6: production-clause($L(\pi)$) ;
7: **fin tant que**
8: choisir littéral ;
9: **fin tant que**
10: **si** pas de conflit **alors**
11: $E = L(\pi)_I$ est une extension ;
12: $E$= complétion(E) ;
13: Condition-discriminante($E$)
14: **sinon**
15: backtrack
16: **fin si**
17: **jusqu'à**

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 $c_i \in C_{STB}$ en utilisant la proposition 4.

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 $E = L(\pi)_I$ est obtenue. Il ne reste plus qu'à effectuer la phase de complétion, ensuite vérifier si l' extension obtenue satisfait la condition dite discriminante pour induire un modèle stable. Le pseudo-code du schéma général de la méthode est donné dans l'algorithme 1.

**Algorithm 2** Procédure propagation-unitaire

**Entrées** La forme clausale $L(\pi)$ du programme $\pi$, la liste de clauses unitaires *Lmonos*, l'interprétation partielle courante $I$
**Sorties** Une interprétation étendue I ou la détection d'un conflit,

1: **tant que** ($Lmonos = \emptyset$) et non (conflit) **Faire**
2: $v \leftarrow next(Lmonos)$ ;
3: $I \leftarrow I \cup \{v\}$ ;
4: $L(\pi) \leftarrow L(\pi) \backslash \{c_i, v \in c_i\}$ ;
5: $c_i \leftarrow c_i \backslash \{\neg v, \neg v \in c_i\}$ ;
6: **si** $longueur(c_i) == 1$ **alors**
7: $Lmonos \leftarrow Lmonos \cup \{c_i\}$
8: **fin si**
9: $Lmonos = Lmonos \backslash \{v\}$ ;
10: $v' \leftarrow inference(v)$ ;
11: $Lmonos = Lmonos \cup \{v'\}$ ;
12: **fin tant que**
13: Si no(conflit) alors retourner $I$ ;
14: else retourner conflit ;

La procédure de propagation-unitaire (Algorithme 2) prends en entrée la forme clausale $L(\pi)$, la liste de clauses unitaires *Lmonos*, et l'interprétation partielle courante $I$. Elle renvoie au retour, soit une interprétation $I$ étendue par la propagation des mono-littéraux, soit un message de conflit si une clause vide est rencontrée. La procédure commence par satisfaire toutes le clauses où apparaît le mono-littéral $v$ et rajoute $v$ à l'interprétation partielle $I$. Puis, elle réduit les clauses dans lesquelles se trouve l'opposé de $v$ ($\neg v$). Si une clause unitaire est crée, elle est alors rajoutée à la liste *Lmonos* qui représente l'ensemble des mono-littéraux. Si une clause vide est détectée, la procédure signale le conflit.

Ensuite, l'algorithme fait appel à la fonction *inférence* (Algorithme 3) qui implémente certaines règles d'inférences basées sur des propriétés théoriques que nous avons démontrées dans la sous-section précédente. Ces règles d'inférence permettent de réduire l'ensemble des points de choix de l'arbre de recherche. Le littéral $v'$ retourné par la procédure *inférence* sera traité comme un mono-littéral.

**Algorithm 3** Procédure inférence

**Entrées** Un littéral v
**Sorties** Un littéral v'

**si** $v == A_i$ **alors**
  $v' \leftarrow \neg not\ A_i$
**fin si**
**si** $v == \neg A_i$ **alors**
  $v' \leftarrow not\ A_i$
**fin si**
**si** $v == not\ A_i$ **alors**
  $v' \leftarrow \neg A_i$
**fin si**

Enfin, La procédure *production-clauses* utilise la unit résolution pour produire les clauses $c_i \in C_{STB}$ selon la propositon 4.

### 3.3 Complexité de l'algorithme

Si $n$ est le nombre de variables de la forme clausale $L(\pi)$ du programme $\pi$, $k$ le cardinal de son STB et $m$ le nombre de clauses de $L(\pi)$, alors la complexité dans le pire des cas de l'algorithme est de l'ordre de O($knm2^k$).

Quant à la complexité spatial, l'algorithme a l'avantage de travailler à espace constant. Il utilise la forme clausale $L(\pi)$ dont la taille est identique à la taille du programme d'entrée $\pi$. La complexité spatiale est constante, elle est de l'ordre de $O(\mid L(\pi) \mid) = O(\mid \pi \mid)$. Cet algorithme peut être utilisé pour des programmes logiques avec boucles et permet de calculer tous les modèles stables d'un programme général.

## 4 Expérimentation

Nous allons dans ce qui va suivre expérimenter notre première implémentation de la méthode sur des instances du problème de coloriage de graphes générées d'une façon aléatoire. Nous évaluons la performance de l'algorithme par rapport au temps de réponse pour trouver le premier modèle stable ou bien pour prouver qu'il n'existe aucun modèle stable. Nous donnons aussi le nombre de point de choix de l'arbre de recherche car il reflète la taille de l'arbre de recherche.

Il ne s'agit ici que d'une première implémentation de la méthode de base utilisant la nouvelle sémantique. Nous sommes encore loin des performances des méthodes optimisées comme *Clasp* [15]. Cette première implémentation et ces premières expérimentation vont juste servir pour étudier le comportement de la méthode et montrer une nouvelle façon de concevoir un système ASP. La comparaison avec les autres méthodes est nécessaire, nous envisageons donc de la faire après avoir introduit les optimisations nécessaires à la méthode.

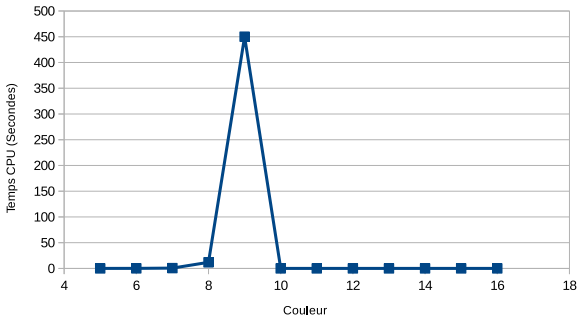


FIGURE 2 – Courbe des temps CPU d'instances de coloration de graphe générés aléatoirement avec n=30 et d=0.5

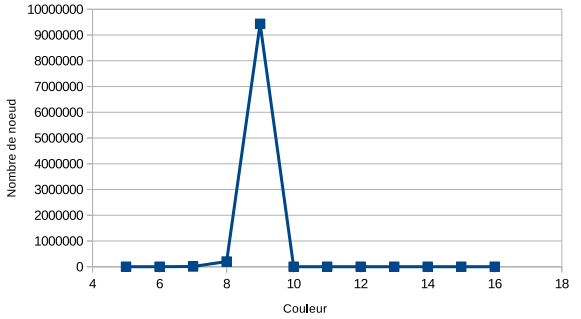


FIGURE 3 – Courbe des nœuds d'instances de coloration de graphe générés aléatoirement avec n=30 et d=0.5

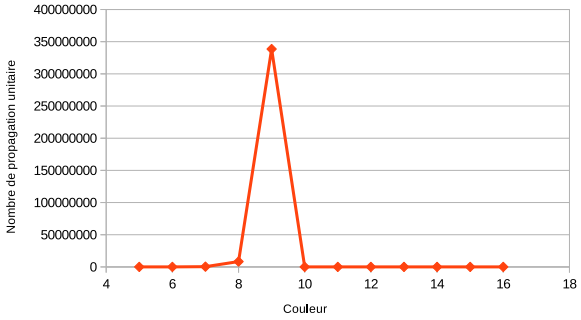


FIGURE 4 – Courbe des propagations d'instances de coloration de graphe générés aléatoirement avec n=30 et d=0.5

Les instances de coloriage de graphes que nous avons testées sont générées à base de trois paramètres d'entrée nécessaires pour le générateur : $n$ le nombre de sommets du graphe, $c$ le nombre de couleurs et $d$ la densité du graphe qui est un nombre entre 0 et 1 exprimant le ratio du nombre d'arêtes dans le graphe sur le nombre de toutes les arêtes possibles.

La figure 2 montre les courbes moyennes représentant les temps CPU pour l'algorithme par rapport à une varia-

| Couleurs | Temps(sec) | Points de choix | Propagations unitaires |
|---|---|---|---|
| 5 | 0,024509 | 239 | 8600 |
| 6 | 0,114816 | 1439 | 52795 |
| 7 | 0,587897 | 10079 | 381716 |
| 8 | 11,816976 | 201599 | 8297582 |
| **9** | **450,134003** | **9434879** | **338575324** |
| 10 | 0,001291 | 94 | 230 |
| 11 | 0,001665 | 126 | 228 |
| 12 | 0,001691 | 156 | 228 |
| 13 | 0,002051 | 186 | 228 |
| 14 | 0,002315 | 216 | 228 |
| 15 | 0,00267 | 246 | 228 |
| 16 | 0,003071 | 276 | 228 |

FIGURE 5 – Résultats détaillés sur des instances de coloriage de graphes générés aléatoirement avec n=30 et d=0.5

tion du nombre de couleurs. On peut observer l'existence d'un pic de difficulté du problème de coloriage de graphes aléatoire. Ce pic se situe en général dans la zone du nombre chromatique (nombre minimal de couleurs pour colorier un graphe). Il correspond à une valeur critique du nombre de couleurs. Cette valeur représente en général le dernier nombre de couleurs avant le nombre chromatique.

La figure 3 et la figure 4 montrent les courbes moyennes représentant le nombre de points de choix ainsi que le nombre de propagations unitaires de l'algorithme par rapport à une variation du nombre de couleurs. Nous pouvons constater une augmentation significative du nombre de propagations unitaires dans la zone du pic de difficulté. Le nombre de points de choix est aussi plus élevé dans la zone du pic de difficulté. Nous pouvons remarquer que la méthode de base arrive à résoudre des instance de coloriage de graphe aléatoires dans la zone difficile. La figure 5 résume les résultats du problème de coloriage de graphes et donne des détailles, la ligne en gras représente le pic de difficulté.

## 5 Conclusion et perspectives

Nous avons étudié dans cet article une méthode de calcul de modèles stables basée sur la nouvelle sémantique [2]. Cette méthode utilise une forme logique clausale ayant une taille identique au programme logique d'entrée. Nous évitons ainsi à la méthode toute lourdeur qu'aurait causée une représentation par la complétion de Clark implémentée dans la plupart des solveurs basés su DPLL. L'autre avantage est que l'énumération est faite sur un sous-ensemble de variables STB représentant le strong backdoor du programme logique d'entrée. Nous avons proposé un ensemble de règles d'inférence qui permettent de réduire le nombre de points de choix et, par conséquent, la taille de l'arbre de recherche. L'algorithme de base a été implémenté et testé sur des instances de coloration de graphes. Les premiers résultats obtenus sont satisfaisants. Notre but dans cette article est de montrer une nouvelle façon de programmer des systèmes ASP. Nous souhaitons avoir les premiers retours sur cette nouvelle méthode.

En perspectives, nous comptons tout d'abord optimiser notre implémentation par l'introduction du matériel utilisé dans les solveurs SAT modernes tel que l'apprentissage de clauses, le redémarrage, les structure paresseuses. Nous envisageons par la suite tester la méthode sur une grande variété de problèmes et la comparer par rapport aux méthodes connues dans le domaine.

Notre approche peut être étendue à d'autres classes de la programmation logique ou à des fragments de logiques non-monotone plus générales telle que la logique des hypothèses [3]. C'est un point que nous envisageons d'étudier dans le futur.

## Références


[1] A Van Gelder, KA Ross, JS Schlipf: *The well-founded semantics for general logic programs*. Journal of the ACM (JACM), 38:619–649, 1991.

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

[3] Camilla Schwind, Pierre Siegel: *A modal logic for hypothesis theory.* Fundamenta Informaticae, 21:89–101, 1994.

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

[5] Esra Erdem, Michael Gelfond, Nicola Leone: *Applications of answer set programming*. AI Magazine, 37:53–68, 2016.

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

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

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

[9] Gilles Audemard, Belaid Benhamou, Pierre Siegel: *Aval: An enumerative method for sat*. Computational Logic—CL 2000, pages 373–383, 2000.

[10] Kaufmann, Benjamin et al: *Grounding and solving in answer set programming*. AI Magazine, 37:25–32, 2016.

[11] Leone, Nicola et al: *The dlv system for knowledge representation and reasoning*. ACM Transactions on Computational Logic (TOCL), 7:499–562, 2006.

[12] Michael Gelfond, Vladimir Lifschitz: *Classical negation in logic programs and disjunctive databases*. New generation computing, 9:365–385, 1991.

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

[14] Ryan Williams, Carla P Gomes, Bart Selman: *Backdoors to typical case complexity*. INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 18:1173–1178, 2003.

[15] Schaub, Torsten et al: *Conflict-driven answer set solving*. IJCAI, 7:386–392, 2007.

[16] Tomi Janhunen, Ilkka Nimelä: *The answer set programming paradigm*. AI Magazine, 37:13–24, 2016.

[17] Victor W. Marek, Miros law Truszczynski: *Stable models and an alternative logic programming paradigm*. The Logic Programming Paradigm, pages 375–398, 1999.

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