Kalpa Publications in Computing

Volume 9, 2018, Pages 58–74

LPAR-22 Workshop and Short Paper Proceedings

Symmetry breaking in a new stable model search method

Tarek Khaled and Belaid Benhamou

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

Abstract

In this work, we investigate the inclusion of symmetry breaking in the answer set programming (ASP) framework. The notion of symmetry is widely studied in various domains. Particularly, in the field of constraint programming, where symmetry breaking made a significant improvement in the performances of many constraint solvers. Usually, combinatorial problems contain a lot of symmetries that could render their resolution difficult for the solvers that do not consider them. Indeed, these symmetries guide the solvers in the useless exploration of symmetric and redundant branches of the search tree. The ASP framework is well-known in knowledge representation and reasoning. However, only few works on symmetry in ASP exist. We propose in this paper a new ASP solver based on a novel semantics that we enhance by symmetry breaking. This method with symmetry elimination is implemented and used for the resolution of a large variety of combinatorial problems. The obtained results are very promising and showcase an advantage when using our method in comparison to other known ASP methods.

1 Introduction

Answer Set Programming (ASP) is an important framework that is used to express and solve a variety of high combinatorial problems, such as graph problems, planning and model checking. ASP is applied in robotics, computational biology and also for industrial purposes. It is an expressive modeling tool, that is able to encode an important number of problems. It has become a popular approach because of the availability of several efficient software tools like *Clasp*[14], *DLV* [18], *Smodels* [25] and other systems based on SAT solver like *ASSAT* [20] and *Cmodels* [19]. The ASP paradigm emerged from the research on knowledge representation and non-monotonic reasoning. Several works have been done to define semantics for logic programs. The main objective behind them is to give a precise meaning to the negation as failure (default negation). The first semantics had been proposed by Clark [8]. This semantics is used in several ASP solvers, but the most known semantics in ASP is the one of stable models [16]. Lately, a new semantics has been proposed [7]. In this semantics, a logic programs is represented by a set of Horn clauses that has the same size as the input *ground* program. The benefit of this semantics is the easy characterization of the stable models and the extension it provides to the stable model semantics.

Here after, we propose a new method for answer set searching that rely on a Boolean enumeration process defined for the ASP paradigm according to the semantics introduced in [7]. This method has the advantage of performing the enumerative process only on a restricted set of literals called the strong backdoor (STB) [26] of the logic program. The search method computes all the possible extensions of

the source logic program from which we can generate all the stable models. It could also compute extra-models corresponding to a kind of extra-extensions that are not captured by the stable model semantics. These extra-models extend the stable model semantics. The stable models are deduced from a sub-set of extensions satisfying what is called here a *a discriminant condition*.

On the other hand, symmetry is studied in several fields including mathematics and artificial intelligence. An object is symmetrical, when the permutation of its elements leaves the object unchanged. Symmetry is a fundamental notion in the satisfiability problem that permit to reduce the computational complexity when dealing with combinatorial problems. The principle of symmetry in propositional logic has been first introduced by Krishnamurthy in [17]. Symmetry has been studied in depth in [4, 5] where a dynamic symmetry elimination is proposed for the DPLL method [11] when solving the satisfiability problem. A static approach that breaks symmetries in a preprocessing phase is introduced in [9]. This static approach consists in adding constraints expressing the *global* symmetry of the initial representation of the problem. This technique has been improved in [2]. Several combinatorial problems when expressed in the ASP paradigm hold a great number of symmetries. For instance the Pigeon Hole problem is known to require exponential time to be solved when the symmetries are not eliminated. Indeed, the ASP solver explores all the symmetrical search spaces. It is possible to avoid exploring these redundant search spaces by breaking the existing symmetries. Until now, only few works on symmetry elimination in ASP have been done. For instance the method presented in [12] treats symmetry in ASP with a statical approach and the encoding is based on the body-atom representation. A different approach is studied in [6]. This last method breaks symmetry in both a statical and a dynamical way.

In this paper, we first start by discussing the new method for ASP solving, then deal with the detection and the elimination of the symmetries of the Horn clausal representation that the new semantics [7] uses to express logic programs. We split the problem of symmetry breaking into three parts. We start by creating a colored graph that represents the Horn clausal form of the given logic program in such a way that the automorphisms of the graph are identical to the symmetries of the Horn clausal representation. Then, a set of generators representing the automorphism group of the graph is computed by using tools like *saucy* [2], *autom* [24] or *nauty* [21]. Finally, symmetry-breaking predicates (SBP) are constructed and added to the Horn clausal formulation. Our approach is inspired by the works presented in [2, 3].

The rest of the paper is organized as follows. First, we recall in Section 2 some notions on logic programming and the semantics [7] on which our search method is based. Then we describe in Section 3 the new ASP search method. In Section 4, we give the definition and the theoretical properties of symmetry. We describe in Section 5 the symmetry detection method before discussing the symmetry-breaking approach in Section 6. Section 7 gives the experimental results obtained on some combinatorial problems. Section 8 concludes the work.

2 Background

We will summarize in the following the notions of permutations, the answer set programming framework and the main theoretical bases of the used semantics [7].

2.1 Permutations

Let $\Omega = \{1, 2, \ldots, N\}$ for some integer N, where each integer might represent a propositional variable or an atom. A permutation of Ω is a bijection mapping defined from Ω into itself. We denote by $Perm(\Omega)$ the set of all permutations of Ω. The pair $(Perm(\Omega), \circ)$ where $\circ$ is the composition of the permutations of $Perm(\Omega)$ forms the permutation group of Ω. The orbit of an element ω on which the group $Perm(\Omega)$ acts is $\omega^{Perm(\Omega)} = \{\omega^\sigma \mid \omega^\sigma = \sigma(\omega), \sigma \in Perm(\Omega)\}$. In other words, the orbit of an element ω under a permutation group are the set of all elements to which the element ω can be mapped.

A *generating set* of the group $Perm(\Omega)$ is a subset *Gen* of $Perm(\Omega)$ such that each element of $Perm(\Omega)$ can be written as a composition of elements of *Gen*. We write $Perm(\Omega)$=< *Gen* >. An element of *Gen* is called a generator. The orbit of $\omega \in \Omega$ can be computed by using only the set of generators *Gen*.

2.2 Answer Set Programming

A logic program π is a finite set of rules of the form *r:head(r)* $\leftarrow$ *body(r)*, where *body(r)* is the set of premises of the rule given as conjunction of literals. The *head(r)* represent the conclusion of the rule which is generally a single literal or in some case a disjunction of literals for disjunctive logic programs. It is in general, given in first order logic and grounders like *gringo* [15] or *lparse* [23] are used to compute all its ground instances. In the sequel, we focus on ground programs, we assume that π is ground.

There are different classes of logic programs. They differ by the presence or the absence of the classical negation and the negation as failure in the rules of the program. A positive logic program π is a set of rules of the form : $r = A_0 \leftarrow A_1, A_2, ..., A_m$, with $(m \geq 0)$ and where A_i is an atom for $0 \leq i \leq m$. There is no classical negation or negation as failure in a positive logic program. A general logic program π is a set of rules of the form : $r = A_0 \leftarrow A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n, (0 \leq m < n)$ where A_i is an atom for $0 \leq i \leq n$ and *not* the symbol expressing the negation as failure. The positive body of r is $body^+(r) = \{A_1, A_2, ..., A_m\}$ and the negative is $body^-(r) = \{A_{m+1}, ..., A_n\}$. The positive projection of r is $r^+ = A_0 \leftarrow A_1, A_2, ..., A_m$. The intuitive meaning of the rule r is the following: if we prove all the atoms of $body^+(r)$ and at the same time no atom of $body^-(r)$ had been proven, then we infer the head A_0. The reduct of a program π with respect to a given set of atoms X is the positive program π^X obtained from π by deleting each rule containing a default-negated atom $not\, A_i$ in its negative body such that $A_i \in X$ and all the atoms $not\, A_j$ from the remaining rules. Formally, $\pi^X = \{r^+ : body^-(r) \cap X = \emptyset\}$. Throughout the rest of the paper, we will focus on general logic programs.

The most known semantics for logic programs is the one of stable models [16]. A set X of atoms is a stable model of π iff X is identical to the minimal Herbrand model of the reduct π^X obtained from π when considering the set of atoms X. This model is also called the canonical model of π^X, it is denoted by $Cn(\pi^X)$. That is, a set X of atoms is a stable model of π if and only if $X = Cn(\pi^X)$. Clark completion semantics [8] is also used to compute the stable models. It is known that each stable model of logic program π is a model of its completion, but a model of the completion is not always a stable model [13] of π. Loop formulas are added to the completion to establish the equivalence between the stable models of π and the models of its Clark completion [20]. The size of the CNF formula resulting from the loop management could vary exponentially within the size of the program. The ASP solvers that use this approach could have an exponential space complexity in the worst case.

2.3 The used semantics

A new semantics is proposed in [7]. This semantics uses a Horn clausal representation to express the considered logic program. This representation has the advantage of having the same size as the one of the input logic program. The new semantics is based on a classical propositional language L composed by two types of literals, a subset of classical literals $V = \{A_i : A_i \in L\}$ and an other subset $nV = \{not\, A_i : not\, A_i \in L\}$. For each literal $A_i \in V$, there is a corresponding literal $not\, A_i \in nV$ designing the negation as failure of L_i. A connection between these two types of literals is expressed by the addition to the propositional language L of an axiom expressing the mutual exclusion between each literal $A_i \in V$ and its corresponding negative literal $not\, A_i \in nV$. This axiom of mutual exclusion is expressed by a set of clauses $ME = \{(\neg A_i \vee \neg not\, A_i) : A_i \in V\}$. A ground general logic program π=$\{r : A_0 \leftarrow$

$A_1, A_2, ..., A_m, not\, A_{m+1}, ..., not\, A_n\}$, $(0 \leq m < n)$ is expressed in the propositional language L by a set of Horn clauses $CR = \{ \bigcup_{r \in \pi} (A_0 \vee \neg A_1 \vee, ..., \neg A_m \vee \neg not\, A_{m+1}, ..., \neg not\, A_n)\}$, $(0 \leq m < n)$ representing all the rules of the logic program to which the set of mutual exclusion clauses $ME = \{(\neg A_i \vee \neg not\, A_i) : L_i \in V\}$ is added. The Horn clausal form of π is the following :

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

The size of the complete representation of the logic program π is approximately equal to $size(\pi) + 2n$, where $n = |V|$. The factor $2n$ corresponds to the set of clauses ME.

The algorithm that we will present in the next section operates on the form $HC(\pi)$. It is an enumerative algorithm that performs on a subset of variables that represent the strong backdoor (STB)[26] of the input logic program. The set STB is formed by the literals of the form $not\, A_i$ that occur in the logic program π.

Definition 1. *Given a logic program π, its strong backdoor is $STB = \{not\, A_i \in nV : \exists r \in \pi, not\, A_i \in body^-(r)\}$.*

The method mainly computes the extensions of $HC(\pi)$ that encodes the stable models. Given a program π and its STB. An extension of $HC(\pi)$ with respect to the STB (or simply an extension of the pair $(HC(\pi), STB)$ is the set of consistent clauses derived from $HC(\pi)$ when adding a maximal set of literals $not\, A_i \in STB$. In other word, an extension of $(HC(\pi), STB))$ is maximally consistent with respect to inclusion of literals $not\, A_i \in STB$, when the addition of one more literal $not\, A_i \in STB$ renders the resulting extension inconsistent. Formally:

Definition 2. *Let $HC(\pi)$ be the Horn CNF encoding of a logic program π, STB its strong backdoor and $S \subseteq STB$. The set $\mathrm{E} = \mathrm{HC}(\pi) \cup \mathrm{S}$ of clauses is then an extension of $(\mathrm{HC}(\pi), \mathrm{STB})$ if the following conditions hold:*

1. E *is consistent,*
2. $\forall \mathrm{not\, A_i} \in \mathrm{STB} - \mathrm{S}, \mathrm{E} \cup \{\mathrm{not\, A_i}\}$ *is inconsistent.*

It is shown in [7], that each stable model of a logic program π corresponds to an extension E of $HC(\pi)$ that satisfies the discriminant condition $(\forall A_i \in V,\ E \models \neg not\, A_i \Rightarrow E \models A_i)$ and vice-versa. There is a one to one bijection between the stable models of π and these extensions. The main proved theoretical properties are the following:

Theorem 1. *If E is an extension of $(HC(\pi), STB)$, that verify the discriminant condition: $\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i$, then $\mathrm{X} = \{\mathrm{A_i} : \mathrm{E} \models \mathrm{A_i}\}$ is a stable model of π.*

It is also shown in [7] that for each stable model of the logic program there exists an extension of $(HC(\pi), STB)$ from which it could be deduced.

Theorem 2. *If X is a stable model of a logic program π, then there exist an extension E of $(L(\pi), STB)$ such that $X = \{L_i \in V : E \models L_i\}$ and which verifies the discriminant condition $(\forall L_i \in V, E \models$ [1] $\neg not L_i \Rightarrow E \models L_i)$.*

Example 1. *Consider the logic program $\pi = \{\ a \leftarrow not\, b; b \leftarrow not\, a\ \}$ The Horn clausal representation of the logic program π is formed by the set $HC(\pi) = CR \cup ME$ where $CR = \{a \vee \neg not\, b, b \vee \neg not\, a\}$, $ME = \{\neg a \vee \neg not\, a, \neg b \vee \neg not\, b\}$ and its strong backdoor is $STB = \{not\, a, not\, b\}$. We can see that $(HC(\pi), STB)$ admits two extensions $E_1 = HC(\pi) \cup \{not\, a\}$ and $E_2 = HC(\pi) \cup \{not\, b\}$. Indeed, E_1 and E_2 are maximally consistent with respect to the set STB. The two extensions satisfy the discriminant condition. Thus, the logic program has two stable models $M_1 = \{b\}$ and $M_2 = \{a\}$ that are deduced from E_1 resp. E_2 by unit resolution.*

[1] The symbol $\models$ expresses the classical logical infrence

3 The proposed search method

We describe here the new search method for stable models that is based on the semantics summarized previously [7]. For a given logic program π, this method computes all the extensions of $(HC(\pi), STB)$ from which the stable models are deduced by unit resolution. Intuitively, the search of the extensions of $(HC(\pi), STB)$ is done by the progressive addition of literals $not\,A_i$ of the STB to $HC(\pi)$ and checking the consistency of the obtained set at each node. If we focus only on stable models, then we just have to look after the extensions verifying the discriminant condition. In other words, we prune the search tree to remove the extra-extensions which don't verify that condition. The proposed method is able to compute all the stable models of a given logic program and in general could search extra-models when stable models do not exist. However, in this work, we limited the search to only stable models. We did this in order to have a safe comparison with other ASP systems that consider only stable models. The enumeration process builds incrementally an extension by alternating in the search tree between deterministic nodes corresponding to the unit propagations and non deterministic nodes that are the choice points. The choice points are defined by the affectation of truth values (true or false) to some literals of the strong backdoor set STB. Some new inference rules that the method uses to increase the number of unit propagations and then reduces the search space are introduced.

3.1 The theoretical bases of the method

We will now introduce some inference rules that the method will use thereafter in the enumeration process.

Definition 3. *Let π be a logic program and $HC(\pi)$ its Horn clausal representation. We define on $HC(\pi)$ two inference rules : $\dfrac{A_i}{\neg not\,A_i}$ and $\dfrac{not\,A_i}{\neg A_i}$.*

These two inference rules represent an efficient implementation of the set of clauses $ME = \bigcup_{A_i \in V} \{(\neg A_i \vee \neg not\,A_i)\}$ of $HC(\pi)$ expressing the mutual exclusion between each pair of atoms A_i and $not\,A_i$.

In the case of our method, the enumeration is done only on the subset of STB literals. Let $C_{STB} = \{c_i = \neg not\,A_{i_1} \vee, ..., \vee \neg not\,A_{i_k} / \mid c_i \mid \geq 1, \forall j \in \{1..k\}, not\,A_{i_j} \in STB\}$ be the set of all possible negative clauses formed by some literals of the set STB and which have at least one literal. The non deterministic treatment of a choice point corresponding to a strong backdoor literal $not\,A_j$ is done by first its assignment to the value $true$ to favor the current extension maximality. The exploration of the branch corresponding to the assignment of the truth value $false$ to $not\,A_j$ is necessary only when the first branch produced at least one sub clause $c_i \in C_{STB}$. This property, could considerably reduce the complexity of the studied method, we will prove it in Proposition 1.

Proposition 1. *Let π be a logic program, $HC(\pi)$ its Horn clausal form, $HC(\pi)_I$ its Horn clausal form simplified by the partial interpretation I corresponding to the current node n of the search tree, $STB = \{not\,A_i : \exists r \in \pi, not\,A_i \in r\}$ the strong backdoor of π, and C_{STB} the set of all possible negative clauses formed by literals of the set STB. If $not\,A_j \in STB$ is the current literal to assign at the node n and the condition $\forall c_i \in C_{STB}$, $HC(\pi)_I \wedge not\,A_j \not\models c_i$ holds, then each extension of $HC(\pi)_I \wedge \neg not\,A_j$ is also an extension of $HC(\pi)_I \wedge not\,A_j$.*

Proof. The subset of clauses $HC(\pi)_I$ corresponding to the current node n of the search tree is the simplified clause system obtained from $HC(\pi)$ by the consideration of the literals that are assigned in the partial interpretation I. By the hypothesis, the literal $not\,A_j$ is the next element of STB that will be assigned at the node n. The set $HC(\pi)_I$ has two types of clauses : the subset of clauses of the form $\neg not\,A_j \vee C_1$ containing the literal $\neg not\,A_j$ and

where C_1 represent a set of pieces of clauses, and the subset of clauses C_2 that do not contain the literal $\neg not\, A_j$. Let $e = not\, A_{i_1} \wedge ... \wedge not\, A_{i_k}$ be an extension of $HC(\pi)_I \wedge \neg not\, A_j$ where $not\, A_{i_j} \in STB$. We shall prove that e is also an extension of $HC(\pi)_I \wedge not\, A_j$. We can see that $HC(\pi)_I \wedge \neg not\, A_j \equiv C_2$ and $HC(\pi)_I \wedge not\, A_j \equiv C_1 \wedge C_2$. The set e is an extension of $HC(\pi)_I \wedge \neg not\, A_j$, thus $C_2 \wedge e$ is consistent. To show that e is also an extension of $HC(\pi)_I \wedge not\, A_j$, it is sufficient to prove that $C_1 \wedge C_2 \wedge e$ is consistent. We proceed by contradiction. That is, by supposing that $C_1 \wedge C_2 \wedge e$ is inconsistent. It results that $C_1 \wedge C_2 \wedge e \models \square$ and thus $C_1 \wedge C_2 \models \neg e$. This means that $C_1 \wedge C_2 \models \neg not\, A_{i_1} \vee ... \vee \neg not\, Ai_k \in C_{STB}$. Therefore, we have $HC(\pi)_I \wedge not\, A_j \models \neg not\, A_{i_1} \vee ... \vee \neg not\, Ai_k \in C_{STB}$. Thus, $HC(\pi)_I \wedge not\, A_j \models c_i \in C_{STB}$ and this contradicts the assumption □

In other words, if no clause $c_i \in C_{STB}$ had been produced at a choice point of the search tree where the value *true* is assigned to a literal $not\, A_j \in STB$, then there is no need to explore the branch corresponding to the negative literal $\neg not\, A_j$. This could avoid to the method to explore redundant and pointless branches.

Proposition 2. *If $HC(\pi)$ is the clausal representation of a logical program π and I the current partial interpretation, then the unit resolution is sufficient to produce from $HC(\pi)_I$ any clause $c_i = \neg not\, A_{i_1} \vee ... \vee \neg not\, Ai_k \in C_{STB}$.*

Proof. The automatic deduction theorem states that proving $HC(\pi)_I \models c_i$ is equivalent to $HC(\pi)_I \wedge \neg c_i \models \perp$. Since $HC(\pi)$ is a set of Horn clauses, it follows that the simplified set of clauses $HC(\pi)_I \wedge \neg c_i$ is also of Horn. That is, $HC(\pi)_I \wedge not\, Ai_1 \wedge ... \wedge not\, Ai_k$ is a set of Horn clauses. Since the unit resolution is sufficient to decide the consistency of any set of Horn clauses, then it is in particularly true for $HC(\pi)_I \wedge \neg c_i$. In other words, the unit resolution is sufficient to show $HC(\pi)_I \wedge \neg c_i \models \perp$ and therefore sufficient to show $HC(\pi)_I \models c_i$ □

To apply the cut induced by Proposition 1 at a given choice point of the search tree, our method must prove that no sub-clause $c_i \in C_{STB}$ is produced at that node. To do this, the method tries to produce such a clause by unit resolution (Proposition 2).

Now, we will show how to exploit the apparition of negative pure (monotone) literals during the search. These literals are frequently overlooked in the implementation of SAT solvers based on DPLL, but for the ASP solvers, they perform a critical role.

Proposition 3. *Let π be a logic program, $HC(\pi)$ its Horn clausal representation and $\neg A_i$ a pure literal of $HC(\pi)$, if X is a stable model of π then $\neg A_i \in X$.*

Proof. The literal $\neg A_i$ is pure in $HC(\pi)$. This means that A_i has no occurrence in $HC(\pi)$, thus the literal A_i can never be inferred. Therefore, A_i can never be a part of a stable model X. By applying the closed world assumption, we have $\neg A_i \in X$ □

This proposition allows to propagate pure negative literals such as mono-literals. Such propagations contribute to the reduction of the number of choice points in the search tree.

Proposition 4. *Let π be a logic program and $HC(\pi)$ its clausal representation, if $\neg A_i$ is true in a stable model X of π then $not\, A_i$ must be true in X.*

Proof. If $\neg A_i$ is true in the stable model X, then the only case where $\neg not\, A_i$ can be produced is the existence of a sub-clause $A_i \vee \neg not\, A_i$ of $HC(\pi)_X$. But in this case, the corresponding extension to X does not verify the discriminant condition. Hence, X would not be a stable model, and this contradicts the assumption □

The previous proposition defines an inference rule ($\frac{\neg A_i}{not\, A_i}$) that the method will use to prune the search tree.

Proposition 5. *Let π be a logic program and $HC(\pi)$ its corresponding clausal form, if $\neg not\, A_i$ is true in a partial model X of π and A_i is false, then X can not be extended to stable model.*

Proof. If $\neg not\, A_i$ is true in the partial model X and at the same time A_i is false in it, then the corresponding extension to X does not verify the discriminant condition. Hence, X can not be extended to a stable model of π □

All the propositions mentioned above bring cuts in the search tree and reduce considerably the search space.

3.2 The algorithm description

In the following, we present the new search algorithm for stable models. Its enumerative process explores a boolean tree search. It looks like the one of a DPLL [11] procedures that is adapted to the ASP framework and to the used semantics [7]. We implemented all the inferences rules introduced previously, to boost the method. The main search process alternates between deterministic unit propagation phases and non deterministic choice point phases where STB clause production is launched on the first branch where a literal $not\,A_i$ of the STB is interpreted to the value $true$. That is, if no clause ($c_i \in C_{STB}$) is produced, then the branch assigning the value false to $not\,A_i$ is not explored. The production process is useless on the last branch. Throughout the two alternate phases, the algorithm affect truth values to literals and develops a similar tree search as the one of a DPLL procedure. If a conflict is encountered during the search, then the algorithm explores the second branch corresponding to the second truth value of the literal representing the current choice point only if a clause $c_i \in C_{STB}$ is produced. otherwise a backtrack is done. An extension candidate is founded either when all the clauses are satisfied, or when all the literals of STB are affected without falsifying any clause. In both cases, the algorithm execute a completing phase that consists in completing the current interpretation by assigning the value true to all the remaining literals $not\,A_i$ of nV and by assigning the value false to all the others literals $A_i \in V$ not assigned yet according to the closed world assumption.

Algorithm 1 The general schema of the new search method

Require: The clausal form $HC(\pi)$ of a logic program π
Ensure: The set S of all the stable models of π
1: $S = \emptyset$
2: **repeat**
3: **while** $STB \neq \emptyset$ and 'no conflict' **do**
4: **while** $Lmonos \neq \emptyset$ or $Lpure \neq \emptyset$ **do**
5: unit-propagation($HC(\pi)$,*Lmonos*,*I*); // propagation of mono-literals
6: inference($HC(\pi)$,*Lpure*,*I*); // propagation of pure-literals
7: clause-production($HC(\pi)$);
8: **end while**
9: literal choice (STB);
10: **end while**
11: **if** no conflict **then**
12: $E = HC(\pi)_I$ // an extension candidate;
13: E= complete(E);
14: **if** Conditions(E) **then**
15: M=PositiveAtoms(E);
16: $S = S \cup M$;
17: **end if**
18: **else**
19: backtrack
20: **end if**
21: **until** All the search space is explored

The algorithm starts by a first call to the unit-propagation procedure to propagate all the mono-literals[2] until the list of mono-literals *Lmono* becomes empty. Then it deals with the pure literals which also could induce mono-literals. When there is no mono-literal and no pure literals to assign, the algorithm tries to produce a clause $c_i \in C_{STB}$ (proposition 2). If we produce a clause $c_i \in C_{STB}$, then the second branch of the current choice point will be explored. Otherwise, if no clause was produced and all the mono-literals and the pure literals are treated, then the second branch of the choice point literal become useless. The enumeration continue by choosing in STB the next literal to assign. This process is repeated either until the satisfaction of all the clauses, or until the assignation of all the literals of

[2] A mono-literal means a unit clause

STB without the appearance of the empty clause. An extension candidate $E = HC(\pi)_I$ is obtained when we reach this state and the completing phase is performed to get a kind of minimal model. After the verification of the maximality and the discriminant conditions on E, a stable model M consisting of the positive atoms A_i of E is extracted and added to the set S. The pseudo-code of the general schema of the method is given in Algorithm 1.

The unit-propagation procedure (Algorithm 2) takes as inputs the clausal form $HC(\pi)$, the list of unit clauses *Lmonos*, and the current partial interpretation I. It returns either an extended interpretation of I or a conflict message if an empty clause is found. The procedure starts by satisfying all the clauses where a certain mono-literal v appears and add v to the partial interpretation I. Then, it reduces the clauses where the opposite of v (the literal $\neg v$) appears. If a unit clause is produced, it will be added to the list of mono-literals *Lmonos*. If an empty clause is detected, the procedure reports the conflict. Secondly, the algorithm calls the Inference function that implements the inference rules seen in the previous subsection. Such inference rules lead to reduce the set of choice points in the search tree. That is, the literal v' returned by the inference procedure will be processed like a mono-literal. Finally, the method calls the clauses-production procedure that uses unit resolution to produce the clauses $c_i \in C_{STB}$ according to Proposition 2.

Algorithm 2 Unit-propagation procedure

Require: the clausal form $HC(\pi)$ of the program π, the list of unit clauses *Lmonos*, the current partial interpretation I
Ensure: An extended interrelation I or a conflict detection,

```
1: while (Lmonos = ∅) and non (conflict) do
2:     v ← next(Lmonos);
3:     I ← I ∪ {v};
4:     HC(π) ← HC(π)\{c_i, v ∈ c_i};
5:     for (c_i ∈ HC(π)) c_i ← c_i\{¬v, ¬v ∈ c_i};
6:     if length(c_i) == 1 then
7:         Lmonos ← Lmonos ∪ {c_i}
8:     end if
9:     Lmonos = Lmonos\{v};
10:    v' ← inference(v);
11:    Lmonos = Lmonos ∪ {v'};
12: end while
13: Si no(conflict) then return I;
14: else return conflict;
```

3.2.1 The algorithm complexity

If n is the number of variables of the clausal form $HC(\pi)$ of the program π, k the cardinality of the set STB and m the number of clauses or $HC(\pi)$, then the algorithm time complexity in the worst case is approximately $O(knm2^k)$. We can notice that the exponential factor of the complexity function depends on the number k representing the size of the strong backdoor set and does not depend on the number of variables n as in the other ASP solvers. The value of k is generally smaller than that one of n, hence a better time complexity.

Unlike the majority of ASP solvers using the Clark completion with the loop management and which have an exponential spatial complexity in the worst case, our method works with constant space. Indeed, the method uses as input the Horn clausal form $HC(\pi)$ whose size is identical to that one of the initial program π and it does not vary during the executions. The spatial complexity is constant, it is of order $O(\mid HC(\pi) \mid) = O(\mid \pi \mid)$ in the worst case.

4 Symmetry definition and properties

The notion of symmetry is widely studied in the field of constraint programming. In this work, we use the clausal encoding $HC(\pi)$ that we augment by the symmetry-breaking predicates that are used to avoid enumerating the symmetrical models or no-goods (interpretations that are not models) of the resulting CNF encoding. We will give in the following the main definitions and properties of the notion of symmetry of a logic program π expressed in its Horn clausal form $HC(\pi)$. First, we define the semantics symmetry:

Definition 4. *Let $HC(\pi)$ be the Horn clausal representation of π and $L_{HC(\pi)}$ its set of literals. A semantics symmetry σ of a $HC(\pi)$ is a permutation defined on the set $L_{HC(\pi)}$, such that $HC(\pi)$ and $\sigma(HC(\pi))$ have the same extensions (the same stable models).*

In other words, a semantics symmetry of the Horn clausal representation of a logic program is a permutation of its literals which preserves the stable models. Now we define the syntactical symmetry:

Definition 5. *Let $HC(\pi)$ be the Horn clausal representation of π and $L_{HC(\pi)}$ its set of literals. A syntactical symmetry σ of a $HC(\pi)$ is a permutation defined on $L_{HC(\pi)}$, such that $HC(\pi) = \sigma(HC(\pi))$.*

A syntactic symmetry of the Horn clausal representation of a logic program is permutation of its literals which leave all the clauses of the program unchanged.

Example 2. *Consider the logic program π of Example 1 where $L_{HC(\pi)} = \{a, b, not\, a, not\, b\}$. The permutation $\sigma = (a,b)(not\, a, not\, b)$ defined on $L_{HC(\pi)}$ is a syntactic symmetry of $HC(\pi)$ since $\sigma(HC(\pi)) = HC(\pi)$.*

Definition 6. *Two literals l and l' of $L_{HC(\pi)}$ are symmetrical if there exists a symmetry σ of $HC(\pi)$ such that $\sigma(l) = l'$.*

Now we define the orbit of a literal:

Definition 7. *Let $HC(\pi)$ be the Horn clausal representation of π, the orbit of a literal $l \in L_{HC(\pi)}$ on which the symmetry group $(Sym(HC(\pi)), \circ)$ acts is $l^{Sym(HC(\pi))} = \{\sigma(l) : \sigma \in Sym(HC(\pi))\}$*

We give below a property that lies between syntactical and semantics symmetry:

Proposition 6. *Each syntactical symmetry of the Horn clausal representation $HC(\pi)$ is a semantics symmetry of $HC(\pi)$.*

Proof. It is trivial to see that a syntactic symmetry of $HC(\pi)$ is always a semantics symmetry of $HC(\pi)$. Indeed, if σ is a syntactic symmetry of $HC(\pi)$, then $\sigma(HC(\pi)) = HC(\pi)$, thus it results that $HC(\pi)$ and $\sigma(HC(\pi))$ have the same stable models □

If E is an extension of $(HC(\pi), STB)$ and σ a syntactic symmetry, we can get another extension of $(HC(\pi), STB)$ by applying σ on the literals which appear in E. Formally:

Proposition 7. *Let σ be a syntactical symmetry of $HC(\pi)$, E is an extension of $(HC(\pi), STB)$ iff $\sigma(E)$ is an extension of $(HC(\pi), STB)$.*

Proof. Suppose that $E = HC(\pi) \cup S'$ is an extension of $(HC(\pi), STB)$. It follows that $\sigma(\mathrm{HC}(\pi)) \cup \sigma(\mathrm{S}')$ is an extension of $(\sigma(HC(\pi)), \sigma(STB))$. We can then deduce that $\mathrm{HC}(\pi) \cup \sigma(\mathrm{S}')$ is an extension of $(HC(\pi), STB)$ since $HC(\pi)$ and STB are invariant under σ. The converse can be shown by considering the converse permutation of σ □

Each stable model of a logic program π corresponds to an extension E of $(HC(\pi), STB)$ that satisfies the discriminant condition $(\forall A_i \in V, E \models \neg not\, A_i \Rightarrow E \models A_i)$. There is a one to one bijection between the stable models of π and these extensions.

Corollary 1. *Let σ be a syntactical symmetry of $HC(\pi)$, I is a stable model of $HC(\pi)$ iff $\sigma(I)$ is a stable model of $HC(\pi)$.*

Example 3. *In Example 2, the orbit of the literal a is $a^{Sym(L_)=\{a,b\}}$ and the one of the literal not a is not $a^{Sym(L_)=\{not\,a,not\,b\}}$. All the literals of a same orbit are all symmetrical. For instance, in Example 1 there are two symmetrical extensions of $(HC(\pi),STB)$. The fist one is $E_1 = HC(\pi) \cup \{not\,a\}$ and the second is $\sigma(E_1) = HC(\pi) \cup \{not\,b\}$. These are symmetrical extensions of $(HC(\pi),STB)$. We can deduce from these two extensions that there are two symmetrical stable models of $HC(\pi)$. The first one is $M_1 = \{b\}$ and the second one is $\sigma(M_1) = \{a\}$.*

If $(Perm(HC(\pi)),\circ)$ denotes the group of permutations of $L_{HC(\pi)}$ and $Sym(L_{HC(\pi)}) \subset Perm(L_{HC(\pi)})$ the subset of permutations of $L_{HC(\pi)}$ that are the syntactic symmetries of $HC(\pi)$, then $(Sym(HC(\pi)),\circ)$ is a sub-group of $(Perm(HC(\pi)),\circ)$ forming the symmetry group of $HC(\pi)$.

5 Symmetry detection

Our symmetry detection is based on graph automorphism search. We first represent the Horn clausal form $HC(\pi)$ by a colored graph $G_{HC(\pi)}$ then compute its set of automorphisms which should be identical to the symmetry group of $HC(\pi)$. This technique has been widely studied in the context of constraint satisfaction and satisfiability problems [9, 2]. Let $G_{HC(\pi)}(V,E)$ be a graph associated to $HC(\pi)$, where V is a set of vertices and $E \subseteq V \times V$ a set of edges. An automorphism (symmetry) of $G_{HC(\pi)}$ is a permutation of the vertices that leaves the graph unchanged. Only the vertices having the same color are permuted together. Given the Horn clausal representation $HC(\pi)$ of the program π, the associated colored graph $G_{HC(\pi)}(V,E)$ of $HC(\pi)$ is defined as follows:

- Each positive literal A_i of $HC(\pi)$ is represented by a vertex $A_i \in V$ of the color 1 in $G_{HC(\pi)}$. The negative literal $\neg A_i$ associated with A_i is also represented by a vertex $\neg A_i$ of color 2 in $G_{HC(\pi)}$. These two vertices are connected by an edge of E in the graph $G_{HC(\pi)}$.
- Each literal $\neg not\,A_i \in STB$ associated with A_i is represented by a vertex $\neg not\,A_i$ of color 3 in $G_{HC(\pi)}$. This vertex is connected to that one representing A_i by an edge of E in the graph $G_{HC(\pi)}$.
- Each literal $\neg not\,A_i \notin STB$ associated with A_i is represented by a vertex $\neg not\,A_i$ of color 4 in $G_{HC(\pi)}$. This vertex is connected to that one representing A_i by an edge of E in the graph $G_{HC(\pi)}$.
- Each clause rule $c_i \in CR$ of $HC(\pi)$ is represented by a vertex $c_i \in CR$ of color 5 in $G_{HC(\pi)}$. An edge connects this vertex c_i to each vertex representing one of its literals.
- Each mutual clause $c_i \in ME$ of $HC(\pi)$ is represented by a vertex $c_i \in V$ of color 6 in $G_{HC(\pi)}$. An edge connects this vertex c_i to the two vertices representing its literals.

This graph construction ensures that only vertices having the same color could be permuted together. The graph $G_{HC(\pi)}$ preserves the syntactic group of symmetries of $HC(\pi)$. That is, the syntactic symmetry group of the representation $HC(\pi)$ of a logic program π is identical to the automorphism group of its graph representation $G_{HC(\pi)}$. The edge between A_i and $\neg A_i$ guarantee that an automorphism that maps A_1 to A_2 also maps $\neg A_1$ to $\neg A_2$. We could use then a graph automorphism system like *saucy*, *autom* or *nauty* to detect the syntactic symmetry group of $HC(\pi)$. These systems return a set of generators of the symmetry group from which we can deduce each symmetry of $HC(\pi)$.

Example 4. *Consider the logic program π of Example 1. The Horn clausal representation of the logic program π is formed by the set $HC(\pi) = CR \cup ME$ where $CR = \{1 : a \vee \neg not\,b, 2 : b \vee \neg not\,a\}$,*

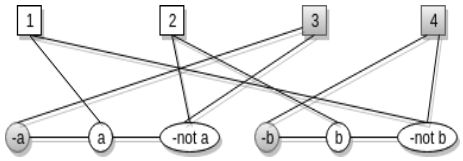

Figure 1: The graph representation of $HC(\pi)$

$ME = \{3 : \neg a \vee \neg not\, a, 4 : \neg b \vee \neg not\, b\}$ and its strong backdoor is given by $STB = \{not\, a, not\, b\}$. Its corresponding graph $G_{HC(\pi)}$ is given in Figure 1. We used in this example five colors that are represented by circles, shaded circles, ellipses, square, and shaded square. The vertices 1,2,3 and 4 represents the fourth clauses of $HC(\pi)$. We can see for instance that the vertex permutation $\sigma = (a,b)(\neg a, \neg b)(\neg not a, \neg not b)(1,2)(3,4)$ is an automorphism of $G_{HC(\pi)}$. The restriction of the automorphism σ to the elements of the STB represents the symmetry $\sigma = (\neg not a, \neg not b)$

6 Symmetry breaking

The different approaches proposed to break symmetries can be classified in two categories: dynamic and static symmetry breaking. The dynamic approach usually looks after and breaks symmetries at each node of the search tree, while the static approach detects and breaks the symmetries in a pretreatment step. In the static approach , symmetries are generally broken by generating additional constraints, called symmetry-breaking predicates (SBP). Here we deal with the static symmetry-breaking technique. The construction of the symmetry-breaking predicates is based on the *lex-leader* method introduced by Crawford et al. in [9] and improved by Aloul et al. [1].

Given a logic program π, the symmetries of its Horn clausal form $HC(\pi)$ induce equivalence classes in the solution/no-good spaces of the program. All the symmetrical interpretations of a stable model/no-good I of π are stable models/no-good of π. Symmetry is then an equivalence relation that defines a partition on the set of interpretations. It is then possible to represent each equivalent class by a representative interpretation. That is, an interpretation I_1 is equivalent to another interpretation I_2 if there exists a symmetry σ of $Sym(HC(\pi))$ such that $I_2 = \sigma(I_1)$. The symmetry-breaking predicates are chosen such that they are true for exactly one interpretation in each equivalent class (the least interpretation in the lex ordering).

Here, we focus only on the STB literals, as they represents the most difficult part in the solutions search. It is sufficient to break the symmetries existing between the STB literals. To do that, we consider an ordering on the literals $not\, A_i$ of the STB and use it to construct a lexicographical order on the set of interpretations. Consider, the symmetry group $Sym(HC(\pi)) = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ of $HC(\pi)$ and a total ordering $not\, A_1 \leq not\, A_2 \leq ... \leq not\, A_n$ on the literals $not\, A_i$ of the STB. The advantage here results in the fact that the size of the STB set is generally smaller than that one of all the literals of $HC(\pi)$. This will lead to get a symmetry elimination predicate of a small size. We obtain the partial *lex-leader* symmetry-breaking predicates *PLL-SBP* by encoding a permutation constraint $PC(\sigma)$ for every permutation σ, defined by:

$$PC(\sigma) = \bigwedge_{1 \leq i \leq n} \left[\bigwedge_{1 \leq j \leq i-1} (not\, A_j = not\, A_j^{\sigma}) \right] \rightarrow (not\, A_i \leq not\, A_i^{\sigma}) \quad (1)$$

The *PLL-SBP* of $HC(\pi)$ is then represented by the conjunction of the permutation constraints asso-

ciated to the considered permutations. It is expressed as follows:

$$PLL-SBP = \bigwedge_{\sigma \in Gen(Sym(HC(\pi))} PC(\sigma) \tag{2}$$

Each $PC(\sigma)$ is translated to a CNF formula whose size is linear in the size n of the STB. The first step is done by introducing the two ordering predicates $l_i = (not\, A_i \leq not\, A_i^\sigma)$ and $g_i = (not\, A_i \geq not\, A_i^\sigma)$. The purpose of adding these two predicates is the elimination of the "equality" and the "less than or equal" operators in predicate $PC(\sigma)$. We obtain the following formula:

$$PC(\sigma) = \bigwedge_{1 \leq i \leq n} \left[\bigwedge_{1 \leq j \leq i-1} l_j g_j \right] \rightarrow l_i \tag{3}$$

More precisely :

$$PC(\sigma) = (1 \rightarrow l_1)(l_1 g_1 \rightarrow l_2)(l_1 g_1 l_2 g_2 \rightarrow l_3) \ldots (l_1 g_1 l_2 g_2 \ldots l_{n-1} g_{n-1} \rightarrow l_n) \tag{4}$$

In the next step, we will use the following known lemma :

Lemma 1. $(a \rightarrow b) \wedge \bigwedge_{i \in I} (abc_i \rightarrow d_i) = (a \rightarrow b) \wedge \bigwedge_{i \in I} (ac_i \rightarrow d_i)$ [3]

Repeated applications of Lemma 1 to the formula 4 leads to a successive elimination of the predicates l_j in the left side of the implications of the formula. We obtain the following:

$$\begin{aligned}
&PC(\sigma) \\
&= (1 \rightarrow l_1)(g_1 \rightarrow l_2)(g_1 l_2 g_2 \rightarrow l_3) \ldots (g_1 l_2 g_2 \ldots l_{n-1} g_{n-1} \rightarrow l_n) \\
&= (1 \rightarrow l_1)(g_1 \rightarrow l_2)(g_1 g_2 \rightarrow l_3) \ldots (g_1 g_2 \ldots l_{n-1} g_{n-1} \rightarrow l_n) \\
&= \cdots \\
&= (1 \rightarrow l_1)(g_1 \rightarrow l_2)(g_1 g_2 \rightarrow l_3) \ldots (g_1 g_2 \ldots g_{n-1} \rightarrow l_n) \\
&= \bigwedge_{1 \leq i \leq n} \left[\bigwedge_{1 \leq j \leq i-1} g_j \right] \rightarrow l_i
\end{aligned}$$

Next, we introduce n auxiliary chaining variables p_i that are defined by:

$$\begin{aligned}
&p_0 = 1 \\
&p_1 = p_0 \wedge g_1 = g_1 \\
&p_2 = p_1 \wedge g_2 = g_1 \wedge g_2 \\
&\ldots \\
&p_{n-1} = p_{n-2} \wedge g_{n-1} = \bigwedge_{1 \leq j \leq n-1} g_j
\end{aligned}$$

The substitution of the order predicates by these definitions gives the following expression:

$$PC(\sigma) = \left[\bigwedge_{1 \leq i \leq n} (p_{i-1} \rightarrow (not\, A_i \leq not\, A_i^\sigma)) \right] \wedge \left[\bigwedge_{1 \leq i \leq n-1} (p_{i-1} \wedge (not\, A_i \geq not\, A_i^\sigma) = p_i) \right] \tag{5}$$

[3] The expression abc_i and ac_i means respectively $a \wedge b \wedge c_i$ and $a \wedge c_i$

which can be converted to a CNF formula consisting of *4n 3-literal* clauses and *n 2-literal* clauses for a total of *14n* literals. A further reduction is possible by replacing the equality predicate of the formula 5 by a one-way implication, which gives rise to the following formula:

$$PC(\sigma) = \left[\bigwedge_{1 \leq i \leq n} (p_{i-1} \rightarrow (not\, A_i \leq not\, A_i^{\sigma})) \right] \wedge \left[\bigwedge_{1 \leq i \leq n-1} (p_{i-1} \wedge (not\, A_i \geq not\, A_i^{\sigma}) \rightarrow p_i) \right] \quad (6)$$

from which, we can obtain a CNF formula that consist of *3n 3-literal* clauses for a total of *9n* literals.

7 Experimentation

Based on the algorithm presented previously, we implemented a new ASP solver that we denote by *HC-asp* to mean Horn Clause ASP. We also included in our system the symmetry breaking to obtain the variant *HC-asp-sym*, the entire system is implemented in C++. The symmetry-breaking method used here, is a static approach that eliminates symmetries in a preprocessing phase. Our ASP system takes as input a ground logic program π produced by the grounder *gringo* [15]. It uses the clausal representation $HC(\pi)$ to compute the stable models of π. The system builds the colored graph $G_{HC(\pi)}$ of $HC(\pi)$ that *saucy* [10] uses to detect the symmetry group of $HC(\pi)$. The symmetry-breaking predicates are computed according to the detected group generators then added to the encoding $HC(\pi)$. After this, an ASP solver could be used on the resulting encoding as a black-box. The general schema of the process is illustrated in Figure 2.

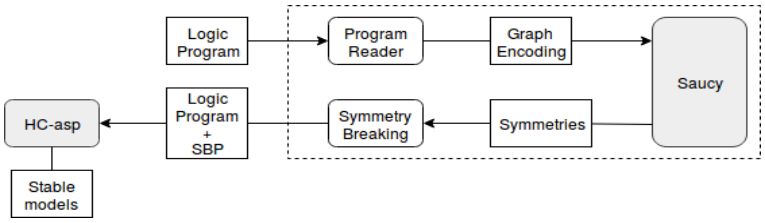

Figure 2: The general schema of ASP solver with the advantage of symmetry

To evaluate our approach, we experimented *HC-asp-sym* on combinatorial problems then compared it performances to that one of *HC-asp* and the ones of *Cmodels* (version 3.86 with *zChaff* as a SAT solver), *Smodels* (version 2.34) and *Clasp*(version 3.3.3). The system *HC-asp-sym* is also compared to the symmetry-breaking system *Sbass* associated to *Clasp*. The system *Sbass* is presented in [12]. This *Sbass* system is based on a static approach that eliminates symmetries in a preprocessing phase. The symmetry-breaking predicates are computed and added to the logic program π. *Clasp* is then applied to the resulting program. The systems are applied to search all the stable models and *gringo* is used as a grounder for all the systems. The programs run on a 4GB Ubuntu (16.10) machine with an Intel Core i5 (1.70GHz x 4). The CPU run time is limited to 24 hours for all the applied systems. The symbol • in the tables means that the corresponding solver fails to solve the instance by the time limit. We report here the results obtained on some known benchmarks that are the Reachability problem, the consistent Pigeon Hole problem, the Ramsey problem, the n-queen problem and the Hamiltonian circuit. The most important task in answer set programming is to enumerate all the stable models of a logic program. symmetry-breaking techniques are also more relevant for calculating all the solutions of a problem. The impact of the symmetry breaking should be very interesting because a solver never explores two points in the search tree that are symmetrical. We chose these benchmarks because of the large• number of

stable models. They are very appropriate to study the behavior of each of the solvers when the number of stable models and the size of the problem increase. We used the same problem encoding for all the solvers.

Table 1: The results obtained on the Reachability, Ramsey, and Pigeon Hole

N°	Instances	#Stable Models					HC-asp-sym				Sbass & Clasp			
			HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#nssmodels	Time	#Sym	#SBP	#nssmodels	Time
1	R_2	1	0.0005	0.0001	0.0002	0.0003	1	6	1	0.0002	1	8	1	0.0001
2	R_3	18	0.0015	0.001	0.0005	0.015	2	23	9	0.0004	2	76	8	0.002
3	R_4	1606	0.030	0.070	0.022	0.091	2	54	725	0.011	2	148	724	0.010
4	R_5	565080	7.12	12.59	7.62	9991.28	3	134	176733	2.94	3	276	120446	3.6
5	R_4_5_5	957	0.011	0.010	0.014	0.049	3	33	66	0.001	4	88	188	0.0001
6	R_4_5_6	27454	0.2	0.5	0.33	18.62	5	72	1023	0.12	5	156	1476	0.04
7	R_4_5_7	1452289	12.64	28.76	18.00	•	5	99	99337	1.52	6	312	25192	0.61
8	R_4_5_8	137578233	1625.14	3329.66	2219.19	•	6	155	3071804	54.67	6	360	12230197	332.51
9	pi4/4	24	0.007	0.009	0.002	0.003	4	84	6	0.0005	6	96	4	0.0001
10	pi5/5	120	0.02	0.02	0.008	0.01	5	147	12	0.0011	8	160	8	0.001
11	pi6/6	720	0.06	0.06	0.04	0.07	6	189	70	0.0048	10	240	54	0.002
12	pi7/7	5040	0.23	0.55	0.30	0.87	7	249	840	0.069	11	336	288	0.036
13	pi8/8	40320	1.65	4.01	2.5	52.34	8	336	2424	0.42	12	448	1800	0.080
14	pi9/9	362880	21.63	47.11	34.60	4591.87	9	702	19634	1.26	13	832	12960	0.51
15	pi10/10	3628800	210.14	494.40	369.80	•	10	846	170354	19.92	14	1008	105840	16.3
16	pi11/11	39916800	2728.53	6936.96	4247.58	•	11	1044	976680	55.77	15	1360	893760	76.25

Tables 1, 2, 3 and 4 give the runtime of the different benchmarks for all the ASP solvers and the number of stable models found by all of them if they finish before the time limit. They also show some symmetry information of the system *HC-asp-sym* and *Sbass & Clasp*: #sym represents the number of detected symmetry, #SBP the number of added SBPs and #nssmodels the number of non symmetrical stable models. The run time is the one of the enumeration of all the solutions including the symmetry preprocessing. The results obtained on the four benchmarks, Reachability(1-4), Pigeon hole(9-16), Ramsey(4-8) are given in Table 1. In general, $HC-asp$ outperforms all the other ASP systems. We can observe that the use of symmetry breaking reduces considerably the solution space for all the benchmarks, and therefore reduces the CPU runtime. This applies to the combination *sbass* and *clasp* and also to our system including the elimination of symmetries. That is, *HC-asp-sym* has better results than those of *HC-asp* and the ones of the other systems. Clearly, we can see that symmetry breaking reduces significantly the number of computed stable models. The gain increases when the problem size and the number of stable models increase.

The Pigeon Hole problem is a good illustration for that observation. For Pigeon Hole of sizes 9 to 11, the solution space is compressed considerably. This reduces the CPU runtime. For the Reachability, and Ramsey problem, we can see that *HC-asp*, *Clasp*, and *Smodels* have comparable results. Again, *HC-asp-sym* get better results on these problems. The comparison of the system *Sbass* associated to *clasp* with *HC-asp-sym* shows that for the problem of Reachability and the Pigeon Hole, both systems have comparable results in terms of CPU time. But for the Ramsey problem, *HC-asp-sym* has better results than *Sbass & Clasp*.

Table 2: The results obtained on the n-queens problem

#Size	#Stable Models						HC-asp-sym				Sbass & Clasp			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Time	#Sym	#SBP	#sol	Time
10	724	0.68	0.23	0.66	0.27	3	468	240	0.10	2	812	278	0.08
11	2680	2.79	1.02	2.95	2.48	3	528	852	0.56	2	936	1032	0.35
12	14200	12.2	8.75	15.19	41.44	3	810	4981	3.69	2	1168	5452	2.7
13	73712	79.55	122.91	87.69	1642.93	3	921	24934	20.53	2	1316	28259	31.14
14	365596	371.73	2631.83	496.98	•	3	918	107371	120.26	2	1588	125728	613.51
15	2279184	2797.02	34337.21	3352.37	•	3	1221	788141	854.31	2	1760	871604	13007.53
16	14772512	12087.40	•	23134.22	•	3	1188	4310853	4976.44	2	2072	•	•
17	95815104	87088.00	•	•	•	3	1275	29227320	29429.41	2	2268	•	•

We also experimented the n-queens and the super n-queens problems. The n-queens problem con-

sists in placing *n* queens on an *nxn* chessboard so that no two queens threaten each other. The super n-queens is a variant of the n-queens where a super queen simultaneously plays the role of a queen and a knight. The obtained results are shown in Table 2 and 3.

Table 3: The results obtained on the super n-queens problem

#Size	#Stable Models					HC-asp-sym				Sbass & Clasp			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Time	#Sym	#SBP	#sol	Time
10	4	0.32	0.01	0.12	0.023	3	467	2	0.013	2	812	2	0.01
11	44	0.54	0.03	0.36	0.049	3	554	13	0.046	2	936	14	0.03
12	156	1.12	0.09	1.65	0.17	3	675	49	0.25	2	1168	63	0.11
13	1876	3.61	0.54	8.12	1.67	3	898	651	1.57	2	1316	651	0.56
14	5180	16.84	5.94	41.53	21.13	3	1086	1960	7.61	2	1588	1767	2.84
15	32516	98.47	122.42	231.52	1241.05	3	1124	11217	38.24	2	1760	11271	39.46
16	202900	653.56	1571.07	23134.22	44433.07	3	1176	62220	207.97	2	2072	77287	654.22
17	1330622	4455.07	48005.13	10001.26	•	3	1435	456661	2313.38	2	2268	503462	18403.85
18	8924976	32649.14	•	77747.20	•	3	1503	2596466	13792.58	2	2620	•	•

We can see in Table 2 that all the methods solved efficiently the small instances of the the n-queens problem (10 to 12 queens) and those of the super n-queens problem (10 to 14 super queens). The results of *HC-asp*, *Clasp* and *Smodels* are comparable on these instances with a slight advantage in the favor of the system *Clasp*. We can observe for both benchmarks that *HC-asp* outperforms drastically all the other methods on the big instances. We can remark that the gain realized by *HP-asp* increases when the size of the problem increases and when the number of solutions increases also. Symmetry breaking greatly contributed to improve the results of *HC-asp*. Again, the number of computed solutions and the time spent on the exploration of all the search space are reduced when applying *HC-asp-sym*. We can see in Table 2 that *sbass* associated with *clasp* has better result than our system on small instances. But from the instance 13, *HC-asp-sym* is more efficient. It can be seen that only the solver *HC-asp-sym* is able to solve all the problems by the time limit. Indeed, *Sbass* associated to *clasp* timed out for the two instances corresponding to 16 and 17 queens. Table 3, shows the results obtained on the super n-queens problem. They look very similar to that ones of the n-queens problem. *HC-asp-sym* is able to solve all the problems by the time limit, *Sbass & clasp* timed out for the instance having 17 super queens. The number of SBPs added in the case of *Sbass* is greater than that one corresponding to the SBPs added by our method. The most plausible hypothesis to explain this observation, is that *saucy* is sensitive to the differences existing in the graph encodings of both approaches. The second fact could be the *lex-leader* of our method that is restricted to the only the STB set. The last benchmark is the Hamiltonian circuit [22]. We computed all the Hamiltonian circuits of some complete oriented graphs whose number of vertices varies from 5 to 12. The behavior of all the solvers are represented in Table 4.

Table 4: The results obtained on the Hamiltonian problem

#Size	#Stable Models					HC-asp-sym				Sbass & Clasp			
		HC-asp	Clasp	Smodels	Cmodels	#Sym	#SBP	#sol	Time	#Sym	#SBP	#sol	Time
5	24	0.012	0.003	0.003	0.003	3	90	6	0.0005	3	108	4	0.00001
6	120	0.026	0.016	0.014	0.02	3	132	48	0.0031	4	224	48	0.0001
7	720	0.09	0.10	0.08	0.12	4	186	288	0.014	3	280	240	0.010
8	5040	0.64	0.78	0.67	1.69	4	246	1992	0.13	6	336	1440	0.11
9	40320	5.76	7.87	6.00	80.75	5	348	5040	0.64	7	392	7920	0.8
10	362880	66.67	89.94	72.25	6177.89	5	396	40320	6.63	8	448	57600	10.91
11	3628800	910	1141.84	964.15	•	6	591	412118	107.26	9	504	786240	123.92
12	39916800	13173.42	22263.37	15964.15	•	6	612	7327392	4604.26	10	560	7781760	4842.33

The results show that all the methods solved efficiently the instances having a number of vertices less than eight (small instances) and *HC-asp* is competitive on the big instances. Again *HC-asp-sym* gets better results than the other methods. The advantage of symmetry is more important when the problem size increases. The comparison with *sbass* associated to *clasp* shows that both systems have comparable results with a slight advantage for our system in terms of CPU time.

8 Conclusion

In this paper, we provided a new method to compute stable models. This method is based on a relatively new semantics and has the advantage of using a Horn clausal logic form whose size is identical to that one of the source ground logic program. It also has a constant spatial complexity. The semantics used prevent the method from the additional burden induced by the the loop management in the semantics of Clark completion, this semantics is used by several ASP solvers. The other benefit of our approach is the simplified enumerative process which is only done on a subset of the variables representing the strong backdoor of the source logic program. This lead to a considerable gain in the time complexity. We also proposed the integration of symmetry breaking in order to avoid exploring isomorphic subspaces. We experimented the proposed method with and without symmetry breaking on a variety of known combinatorial problems. The obtained results showed that our approach is a good alternative to implement ASP solvers and symmetry breaking leads to a significant improvement.

As a future work, we will look first to enhance our implementation with the techniques used in modern SAT solvers such as watched literals, lazy structures, clause learning and restart. This could be beneficial when searching for one stable model like in satisfiability problems. Here we limited the system to search only the stable models of a logic program in order to compare it to other ASP solvers. We are looking to extend the method to search for extra-models that could give a meaning to logical programs in the absence of stable models. Finally, we are looking to investigate some extensions of our approach to others classes of logic programming or to pieces of more general non-monotonic logics.

References

[1] F.A Aloul, , I.L. Markov, and K.A. Sakallah. Efficient symmetry-breaking for boolean satisfiability. *IJCAI*, pages 271–276, 2003.

[2] 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.

[3] 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.

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

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

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

[7] 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.

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

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

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

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

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

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

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

[15] 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.

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

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

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

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

[23] Ilkka Nimelä, Patrik Simons, and Tommi Syrjanen. Smodels: A system for answer set programming. *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, 2000.

[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.