On the Computation of Paracoherent Answer Sets

Giovanni Amendola[1], Carmine Dodaro[1], Wolfgang Faber[2], Nicola Leone[1], Francesco Ricca[1]

[1]Department of Mathematics and Computer Science, University of Calabria, Italy
{amendola,dodaro,leone,ricca}@mat.unical.it
[2]School of Computing and Engineering, University of Huddersfield, United Kingdom
w.faber@hud.ac.uk

Abstract

Answer Set Programming (ASP) is a well-established formalism for nonmonotonic reasoning. An ASP program can have no answer set due to cyclic default negation. In this case, it is not possible to draw any conclusion, even if this is not intended. Recently, several paracoherent semantics have been proposed that address this issue, and several potential applications for these semantics have been identified. However, paracoherent semantics have essentially been inapplicable in practice, due to the lack of efficient algorithms and implementations. In this paper, this lack is addressed, and several different algorithms to compute semi-stable and semi-equilibrium models are proposed and implemented into an answer set solving framework. An empirical performance comparison among the new algorithms on benchmarks from ASP competitions is given as well.

Introduction

In the past decades, many advances in Artificial Intelligence research has been done thanks to studies in the field of knowledge representation and reasoning. Answer Set Programming (ASP) is a premier formalism for non-monotonic reasoning (see, (Brewka, Eiter, and Truszczynski 2011; Gebser et al. 2012)). It is a declarative programming paradigm oriented towards difficult, primarily NP-hard, search problems, which are encoded into a logic program, whose models (answer sets) encode solutions. ASP is based on the stable model (or answer set) semantics introduced by (Gelfond and Lifschitz 1991) and is used to solve complex problems in Artificial Intelligence (Gaggl et al. 2015), Bioinformatics (Campeotto, Dovier, and Pontelli 2015), Databases (Manna, Ricca, and Terracina 2015); Game Theory (Amendola et al. 2016b); more recently ASP has been applied to solve industrial applications (Dodaro et al. 2015; Grasso et al. 2011; Dodaro et al. 2016).

However, a logic program could have no answer set due to cyclic default negation. In this case, it is not possible to draw any conclusion, even if this is not intended. For this reason, theoretical studies have been developed to extend answer set semantics to keep a system responsive in these exceptional cases. To distinguish this situation from reasoning under classical logical contradiction due to strong nega-

tion, called paraconsistent reasoning, it has been referred to it as *paracoherent reasoning* (Amendola et al. 2016a).

In order to deal with this, (Sakama and Inoue 1995) introduced the *semi-stable model semantics* that coincides with answer set semantics whenever a program has some answer set, but admits paracoherent models for each classically consistent program. Recently, (Amendola et al. 2016a) have improved this kind of semantics avoiding some anomalies with respect to basic modal logic properties, resorting to the equilibrium logic (Pearce 2006). Thus, this paracoherent semantics is called *semi-equilibrium model semantics*.[1]

Different possible applications of these paracoherent semantics have been identified, such as debugging, model building, inconsistency-tolerant query answering, diagnosis, planning and reasoning about actions; and computational complexity aspects have been studied (Amendola et al. 2016a). However, to the best of our knowledge, there are no efficient algorithms to compute paracoherent answer sets, obstructing the concrete use of these reasoning approaches. The goal of the paper is to fill this gap, by developing efficient algorithms and their implementation.

In the paper, we consider different algorithms to compute semi-stable and semi-equilibrium models, implementing and integrating them into an answer set building framework. Finally, we report the results of an experimental activity conducted on benchmarks from ASP competitions (Calimeri et al. 2016), identifying the more efficient algorithm.

Preliminaries

We start with recalling answer set semantics, and then present the paracoherent semantics of semi-stable and semi-equilibrium models.

Answer Set Programming

We concentrate on programs over a propositional signature Σ. A *disjunctive rule r* is of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1,...,b_m, not\ b_{m+1},...,not\ b_n, \quad (1)$$

where all a_i and b_j are atoms (from Σ) and $l \geq 0$, $n \geq m \geq 0$ and $l + n > 0$; *not* represents *negation-as-failure*. The set $H(r) = \{a_1,...,a_l\}$ is the *head* of r, while $B^+(r) =$

---

[1]The relationship with other semantics is discussed in the Related Work section.

$\{b_1,...,b_m\}$ and $B^-(r) = \{b_{m+1},...,b_n\}$ are the *positive body* and the *negative body* of r, respectively; the *body* of r is $B(r) = B^+(r) \cup B^-(r)$. We denote by $At(r) = H(r) \cup B(r)$ the set of all atoms occurring in r. A rule r is a *fact*, if $B(r) = \emptyset$ (we then omit $\leftarrow$); a *constraint*, if $H(r) = \emptyset$; *normal*, if $|H(r)| \leq 1$ and *positive*, if $B^-(r) = \emptyset$. A *(disjunctive logic) program* P is a finite set of disjunctive rules. P is called *normal* [resp. *positive*] if each $r \in P$ is normal [resp. positive]. We let $At(P) = \bigcup_{r \in P} At(r)$.

Any set $I \subseteq \Sigma$ is an *interpretation*; it is a *model* of a program P (denoted $I \models P$) iff for each rule $r \in P$, $I \cap H(r) \neq \emptyset$ if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ (denoted $I \models r$). A model M of P is *minimal*, iff no model $M' \subset M$ of P exists. We denote by $MM(P)$ the set of all minimal models of P and by $AS(P)$ the set of all *answer sets (or stable models)* of P, i.e., the set of all interpretations I such that $I \in MM(P^I)$, where P^I is the well-known *Gelfond-Lifschitz reduct* (Gelfond and Lifschitz 1991) of P with respect to I, i.e., the set of rules $a_1 \vee ... \vee a_l \leftarrow b_1,...,b_m$, obtained from rules $r \in P$ of form (1), such that $B^-(r) \cap I = \emptyset$. A program is said to be *coherent* if $AS(P) \neq \emptyset$, *incoherent* otherwise.

Now, we recall a useful extension of the answer set semantics by the notion of *weak constraint* (Buccafurri, Leone, and Rullo 2000). A weak constraint ω is of the form:

$$\leftsquigarrow b_1,\ldots,\ b_m,\ not\, b_{m+1},\ldots,\ not\, b_n.$$

Given a program P and a set of weak constraints W, the semantics of $P \cup W$ extends from the basic case defined above. A constraint $\omega \in W$ is violated by an interpretation I if all positive atoms in ω are true, and all negated atoms are false with respect to I. An *optimum answer set* for $P \cup W$ is an answer set of P that minimizes the number of the violated weak constraints. We denote by $AS^O(P \cup W)$ the set of all optimum answer sets of $P \cup W$.

Paracoherent ASP

Here, we introduce two paracoherent semantics that allow for keeping a system responsive when a logic program has no answer set due to cyclic default negation. These semantics satisfy three desiderata properties identified by (Amendola et al. 2016a).

Semi-Stable Models. Inoue and Sakama (1995) introduced *semi-stable model semantics*. We consider an extended signature $\Sigma^\kappa = \Sigma \cup \{Ka \mid a \in \Sigma\}$. Intuitively, Ka can be read as a is believed to hold. Semantically, we resort to subsets of Σ^κ as interpretations I^κ and the truth values false $\bot$, believed true $\mathbf{bt}$, and true $\mathbf{t}$. The truth value assigned by I^κ to a propositional variable a is defined by

$$I^\kappa(a) = \begin{cases} \mathbf{t} & \text{if } a \in I^\kappa, \\ \mathbf{bt} & \text{if } Ka \in I^\kappa \text{ and } a \notin I^\kappa, \\ \bot & \text{otherwise.} \end{cases}$$

The semi-stable models of a program P are obtained from its *epistemic κ-transformation* P^κ.

Definition 1 (Epistemic κ-transformation P^κ). *Let P be a program. Then its epistemic κ-transformation is defined as*

the program P^κ obtained from P by replacing each rule r of the form (1) in P, such that $B^-(r) \neq \emptyset$, with:

$$\lambda_{r,1} \vee \ldots \vee \lambda_{r,l} \vee Kb_{m+1} \vee \ldots \vee Kb_n \leftarrow b_1,\ldots,b_m, \quad (2)$$
$$a_i \leftarrow \lambda_{r,i}, \quad (3)$$
$$\leftarrow \lambda_{r,i}, b_j, \quad (4)$$
$$\lambda_{r,i} \leftarrow a_i, \lambda_{r,k}, \quad (5)$$

for $1 \leq i, k \leq l$ and $m+1 \leq j \leq n$, where the $\lambda_{r,i}$, $\lambda_{r,k}$ are fresh atoms.

Note that for any program P, its epistemic κ-transformation P^κ is positive. For every interpretation I^κ over $\Sigma' \supseteq \Sigma^\kappa$, let $\mathscr{G}(I^\kappa) = \{Ka \in I^\kappa \mid a \notin I^\kappa\}$ denote the atoms believed true but not assigned true, also referred to as the gap of I^κ. Given a set $\mathscr{F}$ of interpretations over Σ', an interpretation $I^\kappa \in \mathscr{F}$ is *maximal canonical in $\mathscr{F}$*, if no $J^\kappa \in \mathscr{F}$ exists such that $\mathscr{G}(I^\kappa) \supset \mathscr{G}(J^\kappa)$. By $mc(\mathscr{F})$ we denote the set of maximal canonical interpretations in $\mathscr{F}$. Semi-stable models are then defined as *maximal canonical* interpretations among the answer sets of P^κ. Then we can equivalently paraphrase the definition of semi-stable models in (Sakama and Inoue 1995) as follows.

Definition 2 (Semi-stable models). *Let P be a program over Σ. An interpretation I^κ over Σ^κ is a* semi-stable model *of P, if $I^\kappa = S \cap \Sigma^\kappa$ for some maximal canonical answer set S of P^κ. The set of all semi-stable models of P is denoted by $SST(P)$, i.e., $SST(P) = \{S \cap \Sigma^\kappa \mid S \in mc(AS(P^\kappa))\}$.*

Example 1. *Consider the program $P = \{b \leftarrow not\, a;\ c \leftarrow not\, b;\ a \leftarrow c;\ d \leftarrow not\, d\}$. Its epistemic κ-transformation is $P^\kappa = \{\lambda_1 \vee Ka;\ b \leftarrow \lambda_1;\ \leftarrow a, \lambda_1;\ \lambda_1 \leftarrow b, \lambda_1;\ \lambda_2 \vee Kb;\ c \leftarrow \lambda_2;\ \leftarrow b, \lambda_2;\ \lambda_2 \leftarrow c, \lambda_2;\ a \leftarrow c;\ \lambda_3 \vee Kd;\ d \leftarrow \lambda_3;\ \leftarrow d, \lambda_3;\ \lambda_3 \leftarrow d, \lambda_3;\}$, which has the answer sets $M_1 = \{Ka, Kb, Kd\}$, $M_2 = \{\lambda_1, b, Kb, Kd\}$, and $M_3 = \{Ka, \lambda_2, a, c, Kd\}$; as $\mathscr{G}(M_1) = \{Ka, Kb, Kd\}$, $\mathscr{G}(M_2) = \{Kd\}$, and $\mathscr{G}(M_3) = \{Kd\}$. Therefore, among them M_2 and M_3 are maximal canonicals, and hence $M_2 \cap \Sigma^\kappa = \{b, Kb, Kd\}$ and $M_3 \cap \Sigma^\kappa = \{a, c, Ka, Kd\}$ are semi-stable models of P, that also correspond to answer sets of P.*

Semi-Equilibrium Models. Semi-equilibrium models were introduced by (Amendola et al. 2016a) to avoid some anomalies in semi-stable model semantics. Like semi-stable models, semi-equilibrium models may be computed as maximal canonical answer sets, of an extension of the epistemic κ-transformation.

Definition 3 (Epistemic HT-transformation P^{HT}). *Let P be a program over Σ. Then its epistemic HT-transformation P^{HT} is defined as the union of P^κ with the set of rules:*

$$Ka \leftarrow a, \quad (6)$$
$$Ka_1 \vee \ldots \vee Ka_l \vee Kb_{m+1} \vee \ldots \vee Kb_n \leftarrow Kb_1,\ldots,Kb_m, \quad (7)$$

for $a \in \Sigma$, respectively for every rule $r \in P$ of the form (1).

Definition 4 (Semi-equilibrium models). *Let P be a program over Σ, and let I^κ be an interpretation over Σ^κ. Then, $I^\kappa \in SEQ(P)$ if, and only if, $I^\kappa \in \{M \cap \Sigma^\kappa \mid M \in mc(AS(P^{HT}))\}$, where $SEQ(P)$ is the set of semi-equilibrium models of P.*

Algorithm 1: Filtering

1   $M := nextAnswerSet(\Pi, \bot)$; $M^w := M$;
2   $M^w := nextAnswerSet(\Pi, M^w)$;
3   **if** $M^w = \bot$ **then return** M;
4   **if** $gap(M^w) \subset gap(M)$ **then** $M := M^w$;
5   **goto** 2;

Algorithm 2: Guess&Check

1   $M = \bot$;
2   $M = nextAnswerSet(\Pi, M)$;
3   $M^w = nextAnswerSet(\Pi \cup \Pi_M, \bot)$;
4   **if** $M^w = \bot$ **then return** M; **else goto** 2;

Algorithm 3: Minimize

1   $M := nextAnswerSet(\Pi, \bot)$;
2   $\Pi := \Pi \cup \Pi_M$;
3   $M^w := nextAnswerSet(\Pi, \bot)$;
4   **if** $M^w = \bot$ **then return** M; **else** $M := M^w$; **goto** 2;

Algorithm 4: Split

1   $M := nextAnswerSet(\Pi, \bot)$; $C := gap(M)$;
2   **if** $C = \emptyset$ **then return** M;
3   $\Pi := \Pi \cup \Pi_M$; $a := OneOf(C)$;
4   $M^w := nextAnswerSet(\Pi \cup \{\leftarrow a\}, \bot)$;
5   **if** $M^w = \bot$ **then** $\{\Pi := \Pi \cup \{\leftarrow not\, a\}; C := C \setminus \{a\};\}$
6   **else** $\{M := M^w; C := gap(M^w);\}$
7   **goto** 2;

Example 2. *Consider the program P of Example 1. Its epistemic HT-transformation P^{HT} is $P^\kappa \cup \{Ka \leftarrow a;\ Kb \leftarrow b;\ Kc \leftarrow c;\ Kd \leftarrow d;\ Kb \vee Ka;\ Kc \vee Kb;\ Ka \leftarrow Kc;\ Kd \leftarrow Kd\}$, which has the answer sets $\{Ka, Kb, Kd\}$, $\{\lambda_1, b, Kb, Kd\}$, and $\{Ka, \lambda_2, a, c, Kc, Kd\}$. Therefore, the semi-equilibrium models of P are $\{b, Kb, Kd\}$ and $\{a, c, Ka, Kc, Kd\}$.*

In the following, we refer to semi-stable models or semi-equilibrium models as *paracoherent answer sets*.

Complexity Considerations. The complexity of various reasoning tasks with paracoherent answer sets has been analyzed in (Amendola et al. 2016a): while determining the existence of paracoherent answer sets is NP-complete (it is sufficient to test for existence of classical models), paracoherent answer set checking is Π_2^P-complete, leading to Σ_3^P-completeness for brave, and Π_3^P-completeness for cautious reasoning. In this paper, we consider the computation of one paracoherent answer set, which is a functional problem. From previous work it is clear that this task is in $\mathrm{F}\Sigma_3^P$, and actually in $\mathrm{F}\Theta_3^P$ (functional polynomial time with a logarithmic number of calls to a Σ_2^P-complete oracle), because for computing one paracoherent answer set it is sufficient to solve a cardinality-optimization problem.

Computation of a Paracoherent Answer Set

In this section we propose different algorithms to compute one paracoherent answer set. The algorithms take as input a program $\Pi = P^\chi \cup P_g$, where P^χ is a generic epistemic transformation of the ASP program P and P_g is the following set of rules capturing the notion of gap:

$$gap(Ka) \leftarrow Ka, not\, a; \qquad \forall a \in At(P) \qquad (8)$$

Proposition 1. *Let $gap(I) = \{gap(Ka) \mid gap(Ka) \in I\}$, for a set I of atoms. An answer set M of Π is a paracoherent answer set if, and only if, there exists no answer set M_1 of Π such that $gap(M_1) \subset gap(M)$.*

Example 3. *Consider again the program P^κ of Example 1, then Π is the union of P^κ with the following set of rules:*

$$gap(Ka) \leftarrow Ka, not\, a; \quad gap(Kb) \leftarrow Kb, not\, b;$$
$$gap(Kc) \leftarrow Kc, not\, c; \quad gap(Kd) \leftarrow Kd, not\, d;$$

which admits the answer sets $M_1' = M_1 \cup \{gap(Ka), gap(Kb), gap(Kd)\}$, $M_2' = M_2 \cup \{gap(Kd)\}$, and $M_3' = M_3 \cup \{gap(Kd)\}$. Then, $gap(M_1') = \{gap(Ka), gap(Kb), gap(Kd)\}$, $gap(M_2') = gap(M_3') = \{gap(Kd)\}$, thus M_2' and M_3' are paracoherent answer sets.

The output of the algorithms is one semi-stable model of P (if $\chi = \kappa$) or one semi-equilibrium model of P (if $\chi = HT$). In the following, without loss of generality we assume that Π admits at least one paracoherent answer set. In fact, by properties of semi-stable and semi-equilibrium models, this kind of programs admit always a paracoherent answer set (Amendola et al. 2016a).

Moreover, in order to ease the description of the algorithms presented in this section, we introduce the enumeration function *nextAnswerSet*, that takes as input the program Π and an answer set M of Π, and returns as output the next one according to some internal criteria or $\bot$ if no other answer set exists. We abuse of the notation using $M = \bot$ to indicate that the function computes the first answer set.

Filtering. An immediate algorithm for finding a paracoherent answer set is *Filtering*, Algorithm 1. The underlying idea is to enumerate all answer sets of Π and to store the one that is subset-minimal with respect to gap atoms. The algorithm first finds an answer set M of Π. Then, another answer set M^w is searched (line 2). If $gap(M^w)$ is a subset of $gap(M)$ then M is replaced with M^w. Subsequently, the algorithm continues the search until all answer sets have been enumerated. Intuitively, at each step of the computation M is a subset-minimal answer set with respect to the answer sets enumerated so far. Thus, when all answer sets have been enumerated then M is a paracoherent answer set.

Example 4. *Consider again program Π of Example 3. The first call to nextAnswerSet returns M_1' that is stored in M. The second call to nextAnswerSet returns M_2', $gap(M_2')$ is a subset of $gap(M_1')$ therefore M is replaced by M_2'. The third call of nextAnswerSet returns M_3' and M is not modified since $gap(M_3')'$ is not a subset of $gap(M)$. No other answer sets can be enumerated, thus the algorithm terminates returning M.*

The main drawback of Algorithm 1 is that it always computes all answer sets of Π, a potentially exponential number in the size of the atoms of the original program.

In the following we present different algorithms for addressing this inefficiency.

Guess&Check. This algorithm, Algorithm 2, improves Algorithm 1 by reducing the number of computed answer sets. In order to ease the description of the remaining algorithms we introduce the following.

Definition 5. *Given a program Π defined as above. Let M be a model of Π, then Π_M is the following set of constraints:*

$$\leftarrow gap(M); \tag{9}$$
$$\leftarrow gap(Ka); \qquad \forall gap(Ka) \in At(\Pi) \setminus M. \tag{10}$$

Note that (9) contains all atoms in $gap(M)$.

Theorem 1. *Let P be a logic program, let Π be defined as above, and let $M \in AS(\Pi)$. Then, $AS(\Pi \cup \Pi_M) \neq \emptyset$ if, and only if, M is not a paracoherent answer set of P.*

Example 5. *Consider again program Π of Example 3. $\Pi_{M_1'}$ is composed by the following set of constraints:*

$$\leftarrow gap(Ka), gap(Kb), gap(Kd); \quad \leftarrow gap(Kc);$$

whereas $\Pi_{M_2'}$ is composed by the following set of constraints:

$$\leftarrow gap(Kd); \quad \leftarrow gap(Ka);$$
$$\leftarrow gap(Kb); \quad \leftarrow gap(Kc).$$

Note that $AS(\Pi \cup \Pi_{M_1'}) = \{M_2', M_3'\}$ and $AS(\Pi \cup \Pi_{M_2'}) = \emptyset$.

The Guess&Check algorithm finds an answer set M of Π. Subsequently, an answer set of the program $\Pi \cup \Pi_M$ is sought. If such an answer set does not exist then M is a paracoherent answer set and the algorithm terminates returning M. Otherwise, the algorithm iterates the computation until a paracoherent answer set is found.

Example 6. *Consider again program Π of Example 3. The first answer set computed by nextAnswerSet is M_1'. The subsequent check is performed on the program $\Pi \cup \Pi_{M_1'}$, that is coherent. Thus, M_1' is not a paracoherent answer set. Then, nextAnswerSet is called again and it returns M_2'. At this point, $\Pi \cup \Pi_{M_2'}$ is incoherent, therefore the algorithm terminates returning M_2'.*

Algorithm 2 terminates as soon as a paracoherent answer set of P is found. However, in the worst case, it still needs to enumerate all answer sets.

Minimize. The next algorithm is called Minimize and it is reported as Algorithm 3. The idea is to compute an answer set M of Π and then to search for another answer set M^w such that $gap(M^w) \subset gap(M)$. This property is enforced by the constraints of Π_M that are added to the program Π (line 2). If Π admits an answer set, say M^w, then M is replaced by M^w and the algorithm iterates minimizing M. Otherwise, if Π admits no answer set, M is a paracoherent answer set and the algorithm terminates returning M.

Example 7. *Consider again program Π of Example 3. The first answer set computed by nextAnswerSet is M_1'. Thus, the constraints of $\Pi_{M_1'}$ are added to Π. The subsequent check on Π returns an answer set, say M_2', and then Π is modified by adding the constraints of $\Pi_{M_2'}$. At this point, Π is incoherent, therefore the algorithm terminates returning M_2'.*

Algorithm 3 computes at most $|At(P)|$ answer sets.

Split. Another algorithm for computing a paracoherent answer set is called Split, Algorithm 4. The algorithm first computes an answer set M of Π and creates a set C of gap atoms that are included in M. Then, the program Π is modified by adding the constraints of Π_M. Moreover, one of the atoms in C is selected by the procedure $OneOf$, say a. Subsequently, an answer set of $\Pi \cup \{\leftarrow a\}$ is searched. If such an answer set does not exist then a must be included in the paracoherent answer set and thus Π is modified by adding the constraint $\leftarrow not\, a$ and a is removed from the set C. Otherwise, if $\Pi \cup \{\leftarrow a\}$ admits an answer set, say M^w, then M is replaced by M^w and the set C is replaced by the gap atoms that are true in M^w. The algorithm then iterates until the set C is empty, returning M that corresponds to the paracoherent answer set.

Example 8. *Consider again program Π of Example 3. The first answer set computed by nextAnswerSet is M_1'. Thus, C is set to $\{gap(Ka), gap(Kb), gap(Kc)\}$ and the constraints of $\Pi_{M_1'}$ are added to Π. Then, function $OneOf$ selects one of the atoms in C, say $gap(Ka)$. The subsequent check on $\Pi \cup \{\leftarrow gap(Ka)\}$ returns an answer set, say M_2'. Therefore, C is set to $gap(Kd)$ and Π is modified by adding the constraints of $\Pi_{M_2'}$. Then, the function $OneOf$ selects $gap(Kd)$ and the subsequent check on $\Pi \cup \{\leftarrow gap(Kd)\}$ returns $\bot$. Subsequently, Π is modified by adding the constraint $\leftarrow not\, gap(Kd)$ and C is updated by removing $gap(Kd)$. At this point, C is empty, therefore the algorithm terminates returning the latest computed answer set, i.e. M_2'.*

Note that Algorithm 4 requires to compute at most $|At(P)|$ answer sets.

Weak constraints. All the algorithms presented above require the modification of an ASP solver to be implemented. An alternative approach is based on the observation that the gap minimality can be obtained adding to Π the following set of weak constraints, say W:

$$\leftsquigarrow gap(Ka); \qquad \forall a \in At(P). \tag{11}$$

The answer set of the extended program is then an answer set of Π such that a minimal number of weak constraints in W is violated. This means that this answer set that is cardinality minimal with respect to the gap atoms. Therefore, it is also subset minimal with respect to the gap atoms, and so, it is a paracoherent answer set of P.

Theorem 2. *Let P be a program, let Π and W be defined as above. If $M \in AS^O(\Pi \cup W)$, then $M \setminus gap(M)$ is a paracoherent answer set of P.*

Problems	#	Semi-stable semantics					Semi-equilibrium semantics				
		Filt	**G&C**	**Minim**	**Split**	**Weak**	**Filt**	**G&C**	**Minim**	**Split**	**Weak**
KnightTour	26	0	0	0	0	0	0	0	0	0	0
MinDiagn	64	0	37	47	47	0	0	0	8	13	0
QualSpatReas	61	0	26	26	26	10	0	0	0	0	0
StableMarriage	1	0	0	0	0	0	0	0	0	0	0
VisitAll	5	0	5	5	5	5	0	5	5	5	5
Total	**157**	**0**	**68**	**78**	**78**	**15**	**0**	**5**	**13**	**18**	**5**

Table 1: Number of instances solved within the allotted time.

Problems	#	P		$P^{\kappa} \cup P_g$		$P^{HT} \cup P_g$	
		avg #atoms	**avg #rules**	**avg #atoms**	**avg #rules**	**avg #atoms**	**avg #rules**
KnightTour	26	101 480	474 852	358 035	655 709	358 035	1 238 936
MinDiagn	64	238 677	654 412	1 214 042	1 579 409	1 214 042	2 549 729
QualSpatReas	61	11 774	735 317	68 793	789 642	68 793	1 542 607
StableMarriage	1	649 326	110 536 185	-	-	-	-
VisitAll	5	3 820	64 234	13 926	72 776	13 926	140 881

Table 2: Impact of epistemic transformations P^{κ} and P^{HT}.

Note that, the reverse statement does not hold in general. For example, consider the program $P = \{b \leftarrow not\, a;\ c \leftarrow a;\ d \leftarrow b, not\, d\}$. Its semi-equilibrium models are $\{b, Kd\}$ and $\{Ka, Kc\}$. However, $\{Ka, Kc, gap(Ka), gap(Kc)\}$ is not an optimum answer set of $\Pi \cup W$.

Implementation and Experiments

We implemented the algorithms presented in this paper, and we report on an experiment comparing their performance.

Implementation. The computation of a paracoherent answer set is obtained in two steps. First a Java rewriter computes the epistemic transformations P^{κ} and P^{HT} of a propositional ASP program. Then the output of the rewriter is fed in input to a variant of the state-of-the-art ASP solver WASP (Alviano et al. 2015). WASP is an open-source ASP solver, winner of the latest ASP competition (Gebser, Maratea, and Ricca 2015), that we modified by implementing the algorithms presented in the previous section (the source can be downloaded at *https://github.com/alviano/wasp*).

Benchmarks settings. Experiments were run on a Debian Linux system with 2.30GHz Intel Xeon E5-4610 v2 CPUs and 128GB of RAM. Execution time and memory were limited to 1200 seconds and 3 GB, respectively. We use benchmark instances from the latest ASP competition (Gebser, Maratea, and Ricca 2015) collection. We consider all the incoherent instances that do not feature in the encoding neither *aggregates*, nor *choice rules*, nor *weak constraints*, since such features are not currently supported by the paracoherent semantics (Amendola et al. 2016a). This resulted in instances from the following domains: Knight Tour, Minimal Diagnosis, Qualitative Spatial Reasoning, Stable Marriage and Visit All. Instances were grounded with

GRINGO (from *http://potassco.sourceforge.net/*). Grounding times, the same for all compared methods, are not reported.

Results of the experiments. A summary of the result is reported in Table 1, where the number of solved instances for each considered semantics is reported. In the table, Filt is WASP running Algorithm 1, G&C is WASP running Algorithm 2, Minim is WASP running Algorithm 3, Split is WASP running Algorithm 4, and Weak is WASP running the algorithm based on weak constraints.

As a general comment, the algorithm based on the enumeration of answer sets is highly inefficient solving no instances at all. The Guess&Check algorithm outperforms the Filtering algorithm demonstrating that in many cases the enumeration of all answer sets is not needed. The best performing algorithms are Minimize and Split solving both the same number of instances. The performance of the two algorithms are similar also considering the running times. In fact, this is evident by looking at the instance-wise comparison reported in the scatter plot of Figure 1. A point *(x,y)* in the scatter plot is reported for each instance, where *x* is the solving time of the algorithm Minimize whereas *y* is the solving time of the algorithm Split. Concerning the algorithm based on weak constraints, it can be observed that its performance is better than the one of algorithm Filtering. However, it does not reach the efficiency of the algorithms Guess&Check, Minimize and Split.

Concerning the semi-equilibrium semantics, it can be observed that the performance of all algorithms deteriorates. This can be explained by looking at the number of rules introduced by the epistemic HT-transformation, reported in Table 2. In fact, the epistemic HT-transformation introduces approximately twice the number of rules introduced by the epistemic κ-transformation. Moreover, we observe that also in this case the best performing algorithms are Minimize and
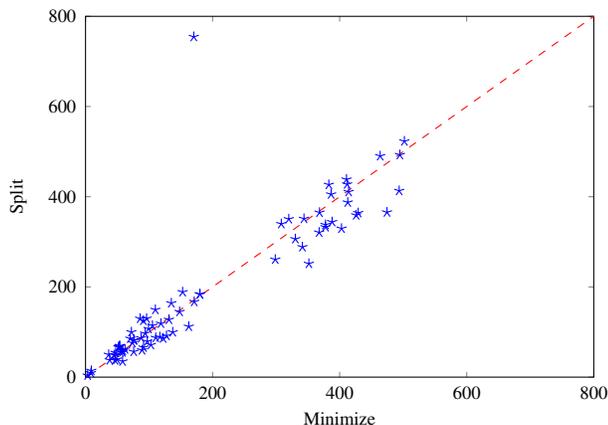
Figure 1: Instance by instance comparison between Minimize and Split on semi-stable semantics.

Split. The latter is slightly more efficient than the former.

Focusing on the performance of the algorithms on the different benchmarks, it can be observed that none of the algorithms was effective on the problems KnightTour and StableMarriage. Concerning KnightTour, we observed that WASP is not able to find any answer set of the epistemic transformations for 12 out of 26 instances. Basically, no algorithm can be effective on such 12 instances, and the remaining ones are hard due to the subsequent checks. Concerning StableMarriage, we observed that java rewriter could not produce the epistemic transformation within the allotted time, because the unique instance of this domain features more than 100 millions rules. The presented algorithms (but Filtering) are able to solve all the considered instances of VisitAll problem, where the epistemic transformations results in a very limited number of atoms and rules (see Table 2).

Related Work

Paracoherent Semantics. Many non-monotonic semantics for logic programs with negation have been proposed that can be considered as paracoherent semantics (Przymusinski 1991; van Gelder, Ross, and Schlipf 1991; Eiter, Leone, and Saccà 1997; Seipel 1997; Balduccini and Gelfond 2003; Pereira and Pinto 2005; Alcântara, Damásio, and Pereira 2005; Galindo, Ramírez, and Carballido 2008; Osorio, Ramírez, and Carballido 2008). However, (Amendola et al. 2016a) have shown that only semi-stable semantics (Sakama and Inoue 1995) and semi-equilibrium semantics (Amendola et al. 2016a) satisfy the following desiderata properties: (i) every consistent answer set of a program corresponds to a paracoherent answer set (*answer set coverage*); (ii) if a program has some (consistent) answer set, then its paracoherent answer sets correspond to answer sets (*congruence*); (iii) if a program has a classical model, then it has a paracoherent answer set (*classical coherence*); (iv) a minimal set of atoms should be undefined (*minimal undefinedness*); (v) every true atom must be derived from the program (*justifiability*).

Computational aspects. Our approach to the computation of paracoherent answer sets is related to the computation of minimal models of propositional theories. The first approaches were proposed for implementing circumscriptive reasoning (cfr. (Gabbay et al. 1994)). Later the attention shifted to the computation of minimal models of first-order clauses (Niemelä 1996; Hasegawa, Fujita, and Koshimura 2000). (Niemelä 1996) proposed a tableaux-based method where candidate models are generated and then tested for minimality. (Hasegawa, Fujita, and Koshimura 2000) proposed a method able to reduce minimality tests on candidate models. The usage of hyperresolution for minimal models of first-order clauses was presented in (Bry and Yahya 2000) and implemented in Prolog. As observed in (Koshimura et al. 2009) these approaches do not take profit of modern non-chronological-backtracking-based solving technology. This limit was overcome in (Koshimura et al. 2009) by an algorithm for computing minimal models of SAT formulas that is based on the same principle as the Minimize algorithm. The computation of minimal models of SAT formulae can be reduced to computing a Minimal Set over a Monotone Predicate (MSMP) (Janota and Marques-Silva 2016). Thus algorithms for MSMP such as those described in (Marques-Silva, Janota, and Belov 2013) could be adapted (by properly taking into account the jump in computational complexity) for computing paracoherent answer sets. Efficient polynomial algorithms for a subclass of positive CNF theories was proposed in (Angiulli et al. 2014). That method cannot be applied directly to find a model of Π that is minimal on the extension of *gap* predicate. The Split algorithm is similar to the algorithms employed for computing cautious consequences of ASP programs (Alviano, Dodaro, and Ricca 2014) and backbones of SAT formulas (Janota, Lynce, and Marques-Silva 2015); nonetheless, to the best of our knowledge, it has no related counterpart in the literature concerning the computation of minimal models.

Conclusion

In this paper, we have tackled the problem of computing paracoherent answer sets, namely semi-stable and stable-equilibrium models, which has not been addressed so far. We have proposed a number of algorithms relying on a program transformation with subsequent calls to an answer set solver. We have conducted an experimental analysis of these algorithms using incoherent programs of the ASP competition, the analysis of incoherent answer set programs being the prime application that we envision for paracoherent answer sets. The experiments show that algorithms Minimize and Split outperform other tested algorithms. The results also show that the computation of a paracoherent answer set is a difficult problem not just theoretically, but also in practice.

Acknowledgements

References

Alcântara, J.; Damásio, C. V.; and Pereira, L. M. 2005. An encompassing framework for paraconsistent logic programs. *J. Applied Logic* 3(1):67–95.

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *LPNMR 2015*, 40–54.

Alviano, M.; Dodaro, C.; and Ricca, F. 2014. Anytime computation of cautious consequences in answer set programming. *TPLP* 14(4-5):755–770.

Amendola, G.; Eiter, T.; Fink, M.; Leone, N.; and Moura, J. 2016a. Semi-equilibrium models for paracoherent answer set programs. *Artif. Intell.* 234:219–271.

Amendola, G.; Greco, G.; Leone, N.; and Veltri, P. 2016b. Modeling and reasoning about NTU games via answer set programming. In *IJCAI 2016*, 38–45.

Angiulli, F.; Ben-Eliyahu, R.; Fassetti, F.; and Palopoli, L. 2014. On the tractability of minimal model computation for some CNF theories. *Artif. Intell.* 210:56–77.

Balduccini, M., and Gelfond, M. 2003. Logic programs with consistency-restoring rules. In *ISLFCR, AAAI 2003 Spring Symposium Series*, 9–18.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Com. ACM* 54(12):92–103.

Bry, F., and Yahya, A. H. 2000. Positive unit hyperresolution tableaux and their application to minimal model generation. *J. Autom. Reasoning* 25(1):35–82.

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.* 12(5):845–860.

Calimeri, F.; Gebser, M.; Maratea, M.; and Ricca, F. 2016. Design and results of the fifth answer set programming competition. *Artif. Intell.* 231:151–181.

Campeotto, F.; Dovier, A.; and Pontelli, E. 2015. A declarative concurrent system for protein structure prediction on GPU. *J. Exp. Theor. Artif. Intell.* 27(5):503–541.

Dodaro, C.; Leone, N.; Nardi, B.; and Ricca, F. 2015. Allotment problem in travel industry: A solution based on ASP. In ten Cate, B., and Mileo, A., eds., *RR 2015*, volume 9209 of *LNCS*, 77–92. Springer.

Dodaro, C.; Gasteiger, P.; Leone, N.; Musitsch, B.; Ricca, F.; and Shchekotykhin, K. 2016. Combining Answer Set Programming and domain heuristics for solving hard industrial problems (Application Paper). *TPLP* 16(5-6):653–669.

Eiter, T.; Leone, N.; and Saccà, D. 1997. On the partial semantics for disjunctive deductive databases. *Ann. Math. Artif. Intell.* 19(1-2):59–96.

Gabbay, D. M.; Hogger, C. J.; Robinson, J. A.; and Siekmann, J. H., eds. 1994. *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume2, Deduction Methodologies*. Oxford University Press.

Gaggl, S.; Manthey, N.; Ronca, A.; Wallner, J.; and Woltran, S. 2015. Improved answer-set programming encodings for abstract argumentation. *TPLP* 15(4-5):434–448.

Galindo, M. J. O.; Ramírez, J. R. A.; and Carballido, J. L. 2008. Logical weak completions of paraconsistent logics. *J. Log. Comput.* 18(6):913–940.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

Gebser, M.; Maratea, M.; and Ricca, F. 2015. The design of the sixth answer set programming competition - report -. In *LPNMR 2015*, 531–544.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput.* 9(3/4):365–386.

Grasso, G.; Leone, N.; Manna, M.; and Ricca, F. 2011. ASP at work: Spin-off and applications of the DLV system. In *Logic Programming, Knowledge Representation, and Non-monotonic Reasoning*, LNCS 6565, 432–451.

Hasegawa, R.; Fujita, H.; and Koshimura, M. 2000. Efficient minimal model generation using branching lemmas. In *CADE-17, 2000*, 184–199.

Janota, M., and Marques-Silva, J. 2016. On the query complexity of selecting minimal sets for monotone predicates. *Artif. Intell.* 233:73–83.

Janota, M.; Lynce, I.; and Marques-Silva, J. 2015. Algorithms for computing backbones of propositional formulae. *AI Commun.* 28(2):161–177.

Koshimura, M.; Nabeshima, H.; Fujita, H.; and Hasegawa, R. 2009. Minimal model generation with respect to an atom set. In *FTP 2009*, CEUR 556.

Manna, M.; Ricca, F.; and Terracina, G. 2015. Taming primary key violations to query large inconsistent data via ASP. *TPLP* 15(4-5):696–710.

Marques-Silva, J.; Janota, M.; and Belov, A. 2013. Minimal sets over monotone predicates in boolean formulae. In *CAV 2013*, 592–607.

Niemelä, I. 1996. A tableau calculus for minimal model reasoning. In *TABLEAUX 1996*, 278–294.

Osorio, M.; Ramírez, J. R. A.; and Carballido, J. L. 2008. Logical weak completions of paraconsistent logics. *J. Log. Comput.* 18(6):913–940.

Pearce, D. 2006. Equilibrium logic. *Ann. Math. Artif. Intell.* 47(1-2):3–41.

Pereira, L. M., and Pinto, A. M. 2005. Revised stable models - a semantics for logic programs. In *EPIA*, 29–42.

Przymusinski, T. C. 1991. Stable semantics for disjunctive programs. *New Generation Comput.* 9(3/4):401–424.

Sakama, C., and Inoue, K. 1995. Paraconsistent stable semantics for extended disjunctive programs. *J. Log. Comput.* 5(3):265–285.

Seipel, D. 1997. Partial evidential stable models for disjunctive deductive databases. In *LPKR*, 66–84.

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38(3):620–650.