Elementary Loops Revisited

Jianmin Ji[a], Hai Wan[b], Peng Xiao[b], Ziwei Huo[b], and Zhanhao Xiao[c]
[a]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
jianmin@ustc.edu.cn
[b]School of Software, Sun Yat-sen University, Guangzhou, China
wanhai@mail.sysu.edu.cn
[c]School of Information Science and Technology, Sun Yat-sen University, Guangzhou, China

Abstract

The notions of loops and loop formulas play an important role in answer set computation. However, there would be an exponential number of loops in the worst case. Gebser and Schaub characterized a subclass *elementary loops* and showed that they are sufficient for selecting answer sets from models of a logic program. This paper proposes an alternative definition of elementary loops and identify a subclass of elementary loops, called *proper loops*. By applying a special form of their loop formulas, proper loops are also sufficient for the SAT-based answer set computation. A polynomial algorithm to recognize a proper loop is given and shows that for certain logic programs, identifying all proper loops of a program is more efficient than that of elementary loops. Furthermore, we prove that, by considering the structure of the positive body-head dependency graph of a program, a large number of loops could be ignored for identifying proper loops. We provide another algorithm for identifying all proper loops of a program. The experiments show that, for certain programs whose dependency graphs consisting of sets of components that are densely connected inside and sparsely connected outside, the new algorithm is more efficient.

Introduction

The notions of loops and loop formulas were first proposed by Lin and Zhao (2004) for normal logic programs. They showed that a set of atoms is an answer set of a program iff it satisfies both the loop formulas and the program, which guarantees the correctness and completeness of SAT-based answer set solvers, like ASSAT (Lin and Zhao 2004), cmodels (Giunchiglia, Lierler, and Maratea 2006), and clasp (Gebser et al. 2007). Besides, the notions and the results have been extended to disjunctive logic programs (Lee and Lifschitz 2003), general logic programs (Ferraris, Lee, and Lifschitz 2006), propositional circumscription (Lee and Lin 2006), and arbitrary first-order formulas with stable model semantics (Lee and Meng 2008).

In general there may be an exponential number of loops (Lifschitz and Razborov 2006). Gebser and Schaub (2005) showed that not all loops are necessary for selecting the answer sets among the models of a program. They introduced the subclass *elementary loops* and refined the Lin-Zhao theorem by considering elementary loops only. In this paper, we revisit the notion of elementary loops and show that some elementary loops could also be disregarded in answer set computation. Rather, we introduce a subclass *proper loops* of elementary loops and further refine the Lin-Zhao theorem by considering a special form of loop formulas only. We show that a proper loop can be recognized in polynomial time and for certain programs, identifying all proper loops of a program is more efficient than that of all elementary loops.

Furthermore, Chen, Ji, and Lin (2013) observed that a preprocessing step applying loop formulas of loops with only one external support rule could significantly improve the computation performances for certain logic programs. This paper shows that, for programs whose dependency graphs consisting of sets of components with densely connected inside and sparsely connected outside, a small number of loop formulas could be chosen using the notion of proper loops and the conjunction of these loop formulas could imply the loop formulas assembled from different components. This result not only explains the observation in (Chen, Ji, and Lin 2013), but also helps us to create an algorithm for identifying all proper loops of a program. Experimental results show that, for these programs, this algorithm is more efficient than directly considering each loop of the program.

Preliminaries

Logic Programs

This paper considers only fully grounded finite normal logic programs. A logic program is a finite set of rules of the form:

$$p_0 \leftarrow p_1, \dots, p_k, not\ q_1, \dots, not\ q_m, \qquad (1)$$

where p_i, $0 \leq i \leq k$, and q_j, $1 \leq j \leq m$, are atoms.

Given a logic program P, let $Atoms(P)$ be the set of atoms in it. For a rule r of the form (1), let $head(r)$ be its head p_0, $body(r)$ the set $\{p_1, \dots, p_k, \neg q_1, \dots, \neg q_m\}$ of literals obtained from the body of the rule with "not" replaced by "$\neg$", $body^+(r)$ the set of atoms in its body, $\{p_1, \dots, p_k\}$, and $body^-(r)$ the set of atoms under not in its body, $\{q_1, \dots, q_m\}$. With a slight abuse of the notion, we use $body(r)$ also for the conjunction of the literals in it. Let R

be a set of rules, we denote $head(R) = \{head(r) \mid r \in R\}$ and $body(R) = \{body(r) \mid r \in R\}$.

Given a rule r of the form (1) and a set S of atoms, we say S *satisfies* $body(r)$, if $body^+(r) \subseteq S$ and $body^-(r) \cap S = \emptyset$. Furthermore, S *satisfies* a logic program P, if for each rule r in P, S satisfies $body(r)$ implies $head(r) \in S$.

Now we define *answer sets* of a program (Gelfond and Lifschitz 1988). Given a logic program P, and a set S of atoms, the Gelfond-Lifschitz transformation of P on S, written P^S, is obtained from P by deleting:

1. each rule having $not\ q$ in its body with $q \in S$, and
2. all negative literals in the bodies of the remaining rules.

For any S, P^S is a set of rules without any negative literals, so that P^S has a unique minimal model, denoted by $\Gamma_P(S)$. Now a set S of atoms is an *answer set* of P iff $S = \Gamma_P(S)$.

Loops and Loop Formulas

It is known that if S is an answer set of P, then S also satisfies P, but the converse may not be true in general. To address this problem, Lin and Zhao (2004) proposed adding so-called *loop formulas* and showed that a set of atoms which satisfies both the program and loop formulas coincides with the answer sets of the program.

To define loop formulas, we have to define loops, defined in terms of positive dependency graphs. Given a logic program P, the *positive dependency graph* of P, written G_P, is the directed graph whose vertices are atoms in P, and there is an arc from p to q if there is a rule $r \in P$ such that $p = head(r)$ and $q \in body^+(r)$. A set L of atoms is said to be a loop of P if for any p and q in L, there is a path from p to q in G_P such that all the vertices in the path are in L, i.e. the L-induced subgraph of G_P is strongly connected. Note that, every singleton whose atom occurs in P is also a loop of P.

Example 1 (Loops) *Consider the logic program P_1:*

$$p \leftarrow . \qquad p \leftarrow r. \qquad q \leftarrow r. \qquad r \leftarrow p. \qquad r \leftarrow q.$$

Figure 1 shows the positive dependency graph of P_1. P_1 has six loops: $\{p\}$, $\{r\}$, $\{q\}$, $\{p, r\}$, $\{r, q\}$, and $\{p, r, q\}$.

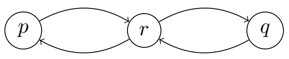

Figure 1: The positive dependency graph of Program P_1

Lin and Zhao defined a formula which says that an atom in a loop cannot be proved by the atoms in the loop only. Thus atoms in the loop can be proved only by using some atoms and rules that are "outside" of the loop. Formally, a rule r is called an *external support* of a loop L if $head(r) \in L$ and $L \cap body^+(r) = \emptyset$. Thus let $R^-(L)$ be the set of external support rules of L. Then the *disjunctive loop formula* of L under P, written $DLF(L, P)$, is the following implication:

$$\bigvee_{p \in L} p \supset \bigvee_{r \in R^-(L)} body(r). \tag{2}$$

An alternative definition of a loop formula proposed in (Lee and Lifschitz 2003) replaces $\bigvee$ in the antecedent of (2) with $\bigwedge$, called the *conjunctive loop formula* and written $CLF(L, P)$:

$$\bigwedge_{p \in L} p \supset \bigvee_{r \in R^-(L)} body(r). \tag{3}$$

Furthermore, we can replace the antecedent of (2) or (3) with a propositional formula that entails $\bigvee_{p \in L} p$ and is entailed by $\bigwedge_{p \in L} p$. For instances, for any loop L, let F_L be a formula formed from atoms in L using conjunctions and disjunctions. Thus let $LF(L, P)$ denote a formula of the form:

$$F_L \supset \bigvee_{r \in R^-(L)} body(r).$$

Theorem 1 (Theorem 1 in (Lee and Lifschitz 2003)) *Let P be a logic program and S a set of atoms. If S satisfies P, then following conditions are equivalent:*

1. *S is an answer set of P;*
2. *S satisfies $DLF(L, P)$ for all loops L of P;*
3. *S satisfies $CLF(L, P)$ for all loops L of P;*
4. *S satisfies $LF(L, P)$ for all loops L of P.*

Lee (2005) proposed that, notions of external support rules and loop formulas can also be defined for sets of atoms.

Elementary Loops

Gebser and Schaub (2005) showed that the Lin-Zhao theorem remains correct even if loop formulas are restricted to a special class of loops called *elementary loops*.

Let X be a set of atoms and Y a subset of X, we say that Y is *outbound* in X for a logic program P if there is a rule r in P such that $head(r) \in Y$, $body^+(r) \cap (X \setminus Y) \neq \emptyset$, and $body^+(r) \cap Y = \emptyset$. Let P be a logic program and L a loop of P, we say that L is an *elementary loop* of P if all nonempty proper subsets of L are outbound in L for P.

Example 1 (Continued) *Program P_1 has six elementary loops: $\{p\}$, $\{r\}$, $\{q\}$, $\{p, r\}$, $\{r, q\}$, and $\{p, r, q\}$.*

Theorem 2 (Theorem 1(d) in (Gebser and Schaub 2005)) *Each of following conditions is equivalent to each of conditions in Theorem 1:*

5. *S satisfies $CLF(L, P)$ for all elementary loops L of P;*
6. *S satisfies $DLF(L, P)$ for all elementary loops L of P;*
7. *S satisfies $LF(L, P)$ for all elementary loops L of P.*

Gebser and Schaub (2005) proved that the problem of deciding whether a given set of atoms is an elementary loop of a logic program is tractable. Let pair (V, E) denote a directed graph where V is the set of vertices and E is the set of edges, i.e., pairs of vertices. Let P be a logic program and X a set of atoms, and we define:

$$EC_P^0(X) = \emptyset,$$

$$EC_P^{i+1}(X) = \{\, (a, b) \mid \text{there is a rule } r \text{ in } P \text{ such that } a = head(r),\ a \in X,\ b \in body^+(r) \cap X, \text{ and all atoms in } body^+(r) \cap X \text{ belongs to the same strongly connected component (SCC) in } (X, EC_P^i(X)) \,\},$$

$$EC_P(X) = \bigcup_{i \geq 0} EC_P^i(X).$$

We call $(X, EC_P(X))$ the *elementary subgraph* of X for P.

Theorem 3 (Theorem 3 in (Gebser and Schaub 2005)) *Let P be a logic program and X a nonempty set of atoms. X is an elementary loop of P iff the elementary subgraph of X for P is strongly connected.*

The above process also provides an algorithm with $O(n^2)$ running time for deciding whether a loop is an elementary loop, where n is number of atoms in the logic program.

An Alternative Definition of Elementary Loops

In this section, we propose an alternative definition of elementary loops. Based on this definition, we provide a new algorithm for deciding whether a loop is an elementary loop. The new algorithm has the same time complexity as Gebser and Schaub (2005), however it follows a top-down strategy while Gebser and Schaub's is a bottom-up approach.

First, we provide a property of conjunctive loop formulas.

Proposition 1 *Let P be a logic program and L_1, L_2 loops of P. If $L_1 \subseteq L_2$ and $R^-(L_1) \subseteq R^-(L_2)$, then $CLF(L_1, P) \supset CLF(L_2, P)$.*

A loop L of a logic program P is called *unsubdued* if there does not exist another loop L' of P such that $L' \subset L$ and $R^-(L') \subseteq R^-(L)$.

Corollary 4 *The following condition is equivalent to each of conditions in Theorem 1:*

5'. S satisfies $CLF(L, P)$ for all unsubdued loops L of P.

Intuitively, a set Y of atoms is outbound in another set X, iff there exists a rule r in P such that $r \in R^-(Y)$ and $r \notin R^-(X)$. Let L be a loop of a program P, all its nonempty proper subsets L' are outbound in L for P iff $R^-(L') \not\subseteq R^-(L)$. Then we have the following proposition.

Proposition 2 *P is a logic program and L is a loop of P. L is an elementary loop of P iff L is an unsubdued loop of P.*

From this definition, we provide Algorithm 1 based on positive dependency graphs for deciding whether a loop L is an elementary loop of a logic program P.

Algorithm 1 ElementaryLoop(L, P)

```
1: for each atom $a \in L$:
2:   $G^* :=$ the $L \setminus \{a\}$ induced subgraph of $G_P$;
3:   $SCC^* :=$ the set of SCCs of $G^*$;
4:   for each $C \in SCC^*$:
5:     if $R^-(C) \subseteq R^-(L)$ then return $C$ else
6:       $G_C :=$ the $C \setminus head(R^-(C) \setminus R^-(L))$ induced subgraph of $G^*$;
7:       $SCC_C :=$ the set of SCCs of $G_C$;
8:       append new elements from $SCC_C$ to $SCC^*$;
9: return $L$
```

Intuitively, ElementaryLoop(L, P) considers sub-loops of L one by one in a top-down process. L' is a sub-loop of L iff L' is a SCC of a subgraph of the L induced subgraph of G_P. For each C of such SCCs, there are two cases: either $R^-(C) \subseteq R^-(L)$ or not. If $R^-(C) \subseteq R^-(L)$, we already find a loop C preventing L to be an elementary loop. If the latter, then for any $r \in R^-(C) \setminus R^-(L)$, $head(r)$ must not be in the sub-loop L' not outbound in L, otherwise r must be in $R^-(L')$, a contradiction with $R^-(L') \subseteq R^-(L)$. So we can remove $head(R^-(C) \setminus R^-(L))$ from the subgraph and continue the procedure.

Note that, the algorithm removes at least one atom in the subgraph at one time. Then, in the worst case, the process runs n^2 times where n is the number atoms in L. As the set of SCCs of a subgraph can be computed in linear time, the time complexity of the algorithm is $O(n^2)$.

Proposition 3 *Let P be a logic program and L a loop of P. The function ElementaryLoop(L, P) returns either L or a set C of atoms such that C is a loop of P, $C \subset L$, and $R^-(C) \subseteq R^-(L)$ in $O(n^2)$, where n is the number of atoms in L. ElementaryLoop(L, P) returns L iff L is an elementary loop of P.*

Proper Loops

This section shows that not all elementary loops are needed for answer set computation. We identify a subclass *proper loops*, and show that, by applying a special form of their loop formulas, they are sufficient for selecting answer sets from models of a logic program.

Let P be a logic program and L a loop of P, we use $RLF(L, P)$ to denote the implication:

$$\bigwedge_{p \in head(R^-(L))} p \supset \bigvee_{r \in R^-(L)} body(r),$$

if $R^-(L) \neq \emptyset$, otherwise

$$\bigwedge_{p \in L} p \supset \bot.$$

Clearly, $RLF(L, P)$ is a special case of $LF(L, P)$. By $RLF(L, P)$, we can generalize the idea of elementary loops.

Proposition 4 *Let P be a logic program and L_1, L_2 loops of P. If $R^-(L_1) \neq \emptyset$ and $R^-(L_1) \subseteq R^-(L_2)$, then $RLF(L_1, P) \supset RLF(L_2, P)$.*

A loop L of a logic program P is called *proper* if there does not exist another loop L' of P such that

- $L' \subset L$ and $R^-(L') \subseteq R^-(L)$, or
- $R^-(L') \neq \emptyset$ and $R^-(L') \subset R^-(L)$.

Theorem 5 *Each of following conditions is equivalent to each of conditions in Theorem 1:*

8. S satisfies $RLF(L, P)$ for all proper loops L of P;

9. S satisfies $DLF(L, P)$ for all proper loops L of P.

When loop formulas are in the form of RLF, more redundant loops can be removed from elementary loops.

Proposition 5 *Let P be a logic program and L a loop of P. If L is a proper loop of P, then L is an elementary loop of P, but not vice versa.*

Example 1 (Continued) *Program P_1 has three proper loops: $\{q\}$, $\{r, q\}$ and $\{p, r, q\}$. $\{p, r\}$ and $\{p\}$ are not proper loops as $R^-(\{p, r\}) = \{p \leftarrow ., r \leftarrow q.\}$, $R^-(\{p\}) = \{p \leftarrow ., p \leftarrow r.\}$ and $R^-(\{p, r, q\}) = \{p \leftarrow .\}$, $\{r\}$ is not a proper loops as $R^-(\{r\}) = \{r \leftarrow p., r \leftarrow q.\}$ and $R^-(\{q, r\}) = \{r \leftarrow p.\}$.*

An elementary loop L is also a proper loop if there does not exist another loop L' such that $R^-(L') \neq \emptyset$ and $R^-(L') \subset R^-(L)$. Note that, L' is not restricted to be a subset of L. Indeed we can restrict the range of possible L's.

Let P be a logic program and S a set of atoms, we say a loop L is a *proper loop of P under S* if $L \subseteq S$ and there does not exist another loop $L' \subseteq S$ such that

- $L' \subset L$ and $R^-(L') \subseteq R^-(L)$, or
- $R^-(L') \neq \emptyset$ and $R^-(L') \subset R^-(L)$.

Now, we provide Algorithm 2 to decide whether a loop L is a proper loop of a program P under a set S.

Algorithm 2 ProperLoop(L, P, S)

```
 1: $G_P^S$ := the $S$ induced subgraph of $G_P$;
 2: $SCC$ := the set of SCCs of $G_P^S$;
 3: for each $C \in SCC$:
 4:     if $C \subset L$ and $R^-(C) \subseteq R^-(L)$ then return $C$
 5:     else if $R^-(C) \neq \emptyset$ and $R^-(C) \subset R^-(L)$ then return $C$
 6:     else if $R^-(C) = \emptyset$ or $R^-(C) = R^-(L)$ then
 7:         for each atom $a \in C$:
 8:             $G^*$ := the $C \setminus \{a\}$ induced subgraph of $G_P^S$;
 9:             $SCC^*$ := the set of SCCs of $G^*$;
10:             append new elements from $SCC^*$ to $SCC$;
11:     else
12:         $G_C$ := the $C \setminus head(R^-(C) \setminus R^-(L))$ induced subgraph
        of $G_P^S$;
13:         $SCC_C$ := the set of SCCs of $G_C$;
14:         append new elements from $SCC_C$ to $SCC$;
15: return $L$
```

Proposition 6 *Let P be a logic program, S a set of atoms, and L a loop of P. ProperLoop(L, P, S) returns L or a set C of atoms such that $C \subseteq S$ is a loop of P and*

- *$C \subset L$ and $R^-(C) \subseteq R^-(L)$, or*
- *$R^-(C) \neq \emptyset$ and $R^-(C) \subset R^-(L)$,*

in $O(n^2)$, where n is the number of atoms in S. ProperLoop(L, P, S) returns L iff L is a proper loop of P under S. Specially, ProperLoop$(L, P, Atoms(P))$ returns L iff L is a proper loop of P.

A native method for computing all proper loops of a program is to use the function ProperLoop to filter out proper loops from every loops of the program. The method for proper loops can be improved by the following proposition.

Proposition 7 *Let P be a logic program and L a proper loop of P such that $R^-(L) \neq \emptyset$. If L' is a loop of P such that $L' \subset L$ and $head(R^-(L)) \subseteq L'$, then L' is not proper.*

Then, we provide Algorithm 3 for computing all proper loops of a program P under a set S. We denote $ploop(P, S)$ the set of proper loops L of P under S below.

Algorithm 3 ProperLoops(P, S)

```
 1: $Loops := \emptyset$;
 2: $G_P^S$ := the $S$ induced subgraph of $G_P$;
 3: $SCC$ := the set of SCCs of $G_P^S$;
 4: for each $C \in SCC$:
 5:     $C^*$ := ProperLoop$(C, P, S)$;
 6:     if $C^* = C$ then
 7:         append $C$ to $Loops$;
 8:         for each $a \in head(R^-(C))$
 9:             $G_C$ := the $C \setminus \{a\}$ induced subgraph of $G_P^S$;
10:             $SCC_C$ := the set of SCCs of $G_C$;
11:             append new elements from $SCC_C$ to $SCC$;
12:     else
13:         for each $a \in C$:
14:             $G_C$ := the $C \setminus \{a\}$ induced subgraph of $G_P^S$;
15:             $SCC_C$ := the set of SCCs of $G_C$;
16:             append new elements from $SCC_C$ to $SCC$;
17: return $Loops$
```

Intuitively, the function ProperLoops(P, S) considers every sub-loops of S except loops that are excluded by Proposition 7 in a top-down process.

Proposition 8 *Let P be a logic program and S a set of atoms. The function ProperLoops(P, S) returns the set of proper loops of P under S (i.e. $ploop(P, S)$).*

Lifschitz and Razborov (2006) proved that exponentially many loop formulas may be necessary for filtering out the program's answer sets, so there would be an exponential number of proper loops in the worst case. However, for some programs, the number of proper loops is much smaller than that of elementary loops and the function ProperLoops$(P, Atoms(P))$ returns all proper loops faster than the native procedure that computes all elementary loops.

For Hamiltonian Circuit (HC) problem[1] (Niemelä 1999), we consider graphs that represent networks consisting of sets of components which are densely connected inside but have only a few connections among them.

These networks are ubiquitous, such as countries consisting of big cities that are connected by only a few highways, cities consisting of populated neighborhoods that are connected by a few "main roads", and circuits that are often composed of components that are highly connected inside but have only a few connections between them.

To simplify things a bit, we model these networks by graphs consisting of some complete subgraphs that are connected by a few arcs between them. Specifically, we consider graphs of the form M-N-K: a graph with M copies of the complete graph with N nodes, $C_1, \ldots, C_M$, and with K arcs from C_i to C_{i+1}, for each $1 \leq i \leq M$ (C_{M+1} is defined to be C_1).

[1]HC problem could carry over to logic programs whose positive dependency graphs have similar structures. Our approach focuses on loops and loop formulas, so the experiment results are the same for all ASP programs whose positive dependency graphs have a similar structure. Furthermore, the structures considered in the experiment occur frequently in practice.

Table 1: Computing Elementary Loops and Proper Loops

Problem	Elementary Loops		Proper Loops	
	number	time	number	time
2-5-1	69	0.13	23	0.04
2-5-2	135	0.12	64	0.06
2-6-1	211	1.95	53	0.18
2-6-2	473	5.22	192	0.75
2-6-3	598	4.45	346	1.29
2-7-1	685	24.88	115	0.91
2-7-2	1734	74.83	616	4.77
2-7-3	2883	46.56	1519	5.92
2-8-1	2399	274.69	241	4.55
2-8-2	6537	162.34	2124	15.74
2-8-3	—	>10min	5628	37.68
3-5-1	95	0.03	35	0.03
3-5-2	161	0.15	76	0.08
3-6-1	268	0.29	80	0.16
3-6-2	532	1.70	219	0.54
3-7-1	—	>10min	173	20.70
3-7-2	—	>10min	7555	162.00
3-7-3	—	>10min	44815	593.16
3-8-1	—	>10min	362	224.26
3-8-2	—	>10min	—	>10min
4-5-1	93	0.08	50	0.05
4-5-2	135	0.25	106	0.13
4-6-1	—	>10min	106	43.30
4-6-2	—	>10min	8364	412.12
4-7-1	—	>10min	—	>10min

Table 1 contains the numbers and running times of elementary loops and proper loops for these HC programs.[2] For each M-N-K entry in the table, we randomly created 20 different such graphs, and the numbers and times reported in the table refers the average numbers and times for the resulting 20 programs. The numbers and times under "Elementary Loops" (resp. "Proper Loops") refers to the numbers of all elementary loops (resp. proper loops) and the run times (in seconds) of the native method for elementary loops (resp. the function ProperLoops). As can be seen, both numbers and running times are less when looking for proper loops.

Separators for Positive Body-Head Dependency Graphs

By considering the positive body-head dependency graph (Linke and Sarsakov 2005) of a program, Proposition 7 can be extended and a larger number of loops could be proved to be not proper. Then we provide an alternative approach for computing all proper loops. For many programs, the new approach is more efficient than ProperLoops.

Given a logic program P, the *positive body-head dependency graph* of P, written G^*_P, is the directed graph whose vertices are elements in the set $Atoms(P) \cup body(P)$, and there are two kinds of arcs in G^*_P, (a, B) and (B, a) where

- (a, B) if there is a rule $r \in P$ such that $a = head(r)$ and $B = body(r)$,
- (B, a) if there is a rule $r \in P$ such that $B = body(r)$ and $a \in body^+(r)$.

[2]Our experiments were done on an AMD A10-5800K (3.8GHz) CPU and 3.3GB RAM. Times are in CPU seconds as reported by Linux "/usr/bin/time" command. For more information, please visit http://ss.sysu.edu.cn/%7ewh/properloop.html

Let C be a set of atoms in P, *the C induced subgraph of G^*_P* is defined as the directed graph whose vertices are elements in the set $C \cup \{body(r) \mid head(r) \in C$ and $body^+(r) \cap C \neq \emptyset\}$, and there are two kinds of arcs in the graph:

- (a, B) if there is a rule $r \in P$ such that $a = head(r)$, $B = body(r)$, $a \in C$, and $body^+(r) \cap C \neq \emptyset$,
- (B, a) if there is a rule $r \in P$ such that $B = body(r)$, $a \in body^+(r)$, $a \in C$, and $head(r) \in C$.

Clearly, L is a loop of P iff the L induced subgraph of G^*_P is strongly connected. Figure 2 presents an example of the positive body-head dependency graph of a program.

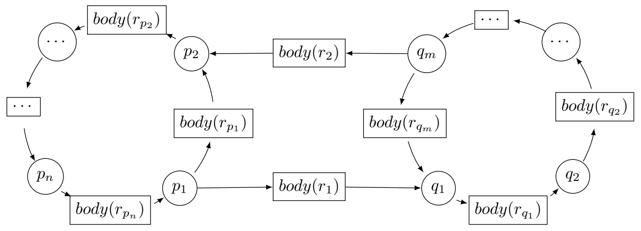

Figure 2: A positive body-head dependency graph

Intuitively, if we can "remove" the vertex $body(r_1)$ or $body(r_2)$ in Figure 2, then the number of loops would be greatly reduced. For instance, let the left subgraph be a strongly connected subgraph with n number of atoms and the right subgraph a strongly connected subgraph with m number of atoms, then the number of loops would be $2^{m+n-4}+2^n+2^m-2$. After "removing" the vertex $body(r_1)$ or $body(r_2)$, the number of loops is reduced to $2^n + 2^m - 2$.

Firstly, we extend the notion of proper loops. Let P be a logic program, S a set of atoms, and R a set of rules, we say a set C of atoms is a *proper set of P under S for R*, if $C \subseteq S$, for each $r \in R$, $body^+(r) \cap C \neq \emptyset$, and there does not exist another set $C' \subseteq S$, for each $r \in R$, $body^+(r) \cap C \neq \emptyset$ s.t.

- $C' \subset C$ and $R^-(C') \subseteq R^-(C)$, or
- $R^-(C') \neq \emptyset$ and $R^-(C') \subset R^-(C)$.

Next, we use $pset(P, S, R)$ to denote the set of proper sets of P under S for R. Note that, we can obtain an algorithm for ProperSets(P, S, R) from ProperLoops(P, S) by:

1. instead of considering each SCCs in a subgraph but considering each possible subsets;
2. only appending a proper set C if for each $r \in R$, $body^+(r) \cap C \neq \emptyset$.

Let S_1 and S_2 be sets of atoms of P such that $S_1 \cap S_2 = \emptyset$, we define $edge(S_1, S_2) = \{r \mid r \in P, head(r) \in S_1$, and $body^+(r) \cap S_2 \neq \emptyset\}$.

Proposition 9 *Let P be a logic program and S_1, S_2 sets of atoms of P such that $S_1 \cap S_2 = \emptyset$. If $L \subseteq S_1 \cup S_2$ is a loop of P, $R^-(L) \neq \emptyset$ such that*

- *$L \cap S_1 \neq \emptyset$ and $L \cap S_1$ is not a proper set of P under S_1 for $edge(L \cap S_2, L \cap S_1)$, or*
- *$L \cap S_2 \neq \emptyset$ and $L \cap S_2$ is not a proper set of P under S_2 for $edge(L \cap S_1, L \cap S_2)$,*

then there exists a set $C \subseteq S_1 \cup S_2$ such that

- *$C \cap S_1 \neq \emptyset$ implies $C \cap S_1$ is a proper set of P under S_1 for $edge(L \cap S_2, L \cap S_1)$,*
- *$C \cap S_2 \neq \emptyset$ implies $C \cap S_2$ is a proper set of P under S_2 for $edge(L \cap S_1, L \cap S_2)$, and*
- *$RLF(C, P) \supset RLF(L, P)$.*

Due to the space limitation, we omit the proof here. From Proposition 9, we get the following theorem.

Theorem 6 *Let P be a logic program, S_1 and S_2 sets of atoms in P with $S_1 \cap S_2 = \emptyset$. If L is a loop of P such that $L \subseteq S_1 \cup S_2$, then*

$$\bigwedge_{L_1 \in ploop(P,S_1)} RLF(L_1, P) \wedge \bigwedge_{L_2 \in ploop(P,S_2)} RLF(L_2, P)$$
$$\wedge \bigwedge_{\substack{R_1 \neq \emptyset,\, R_1 \subseteq edge(S_2,S_1) \\ R_2 \neq \emptyset,\, R_2 \subseteq edge(S_1,S_2) \\ C_1 \in pset(S_1,P,R_1) \\ C_2 \in pset(S_2,P,R_2)}} RLF(C_1 \cup C_2, P) \supset RLF(L, P)$$

Let P be a logic program, L a loop of P, we call a set R of rules a *separator* of P for L, if there exists two sets S_1 and S_2 of atoms such that $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = L$ and $R = edge(S_1, S_2) \cup edge(S_2, S_1)$. A separator R of P for L is *minimal*, if there does not exist another separator R' of P for L such that $|R'| < |R|$. In fact, a minimal separator of P for L can be computed by the function MinimalSeparator(P, L) in Algorithm 4 in a polynomial time.

Algorithm 4 MinimalSeparator(P, L)

```
1: $G_P^S$ := the $S$ induced subgraph of $G_P^*$;
2: $G_1$ := the resulting graph of $G_P^S$ by eliminating vertexes in $Atoms(P)$;
3: $G_2$ := the resulting graph by changing a directed graph $G_1$ to an undirected graph;
4: $G_3$ := the resulting graph of $G_2$ by evaluating each edge with a infinity number and replacing each vertex by two new vertexes with an edge valued with 1 between both vertexes;
5: $Cut$ := a minimum cut of $G_3$ computed by the Stoer-Wagner algorithm;
6: $R$ := the set of corresponding rules for $Cut$;
7: return $R$
```

Now we provide an alternative approach by the function ProperLoops$^*(P, L)$ in Algorithm 5 for identifying all proper loops of a program.

Proposition 10 *Let P be a logic program and L a loop of P. The function ProperLoops$^*(P, L)$ returns the set of proper loops of P under L.*

Given a loop L of P, when the size of the minimal separator is small, it is quite possible that the function ProperLoops$^*(P, L)$ is more efficient than ProperLoops(P, L). For instance, let $|L| = n$, L is participated into S_1 and S_2, $|S_1| = m$, $|edge(S_2, S_1)| = k_1$, for each nonempty subset $R_1 \subseteq edge(S_2, S_1)$, $|\text{ProperSets}(P, S_1, R_1)| = C_1$, $|edge(S_1, S_2)| = k_2$, and for each nonempty subset $R_2 \subseteq edge(S_1, S_2)$, $|\text{ProperSets}(P, S_2, R_2)| = C_2$, then the number of loops that need to be considered in ProperLoops(P, L) is 2^n and the number of loops that need to be considered in ProperLoops$^*(P, L)$ is $2^m + 2^{n-m} + k_1 k_2 C_1 C_2$. In the worst case, $C_1 = 2^m$ and $C_2 = 2^{n-m}$, then ProperLoops$^*(P, L)$ would be less efficient. However, when k_1, k_2, C_1, and C_2 are small, the number would be much smaller than 2^n.

Algorithm 5 ProperLoops$^*(P, L)$

```
1: $Loops := \emptyset$;
2: compute a minimal separator for $L$ which participates $L$ into $S_1$ and $S_2$;
3: append proper loops in ProperLoops$(P, S_1)$ and ProperLoops$(P, S_2)$ to $Loops$;
4: $Pset_1 := \emptyset$ and $Pset_2 := \emptyset$;
5: for each nonempty subset $R_1 \in edge(S_2, S_1)$:
6:     append new loops in ProperSets$(P, S_1, R_1)$ to $Pset_1$;
7: for each nonempty subset $R_2 \in edge(S_1, S_2)$:
8:     append new loops in ProperSets$(P, S_2, R_2)$ to $Pset_2$;
9: for each pair $(C_1, C_2)$ in $Pset_1 \times Pset_2$:
10:    append the set $C_1 \cup C_2$ to $Loops$;
11: return $Loops$
```

Table 2 contains the numbers of checked loops and running times of ProperLoops$(P, Atoms(P))$ and ProperLoops$^*(P, Atoms(P))$ for corresponding HC programs. As can be seen, both numbers and running times are less for the function ProperLoops$^*(P, Atoms(P))$. Note that, for some programs, the number of considered loops for ProperLoops$^*(P, Atoms(P))$ is even less than the number of proper loops, as a large number of these proper loops are constructed from two proper sets identified before.

Table 2: Comparing ProperLoops and ProperLoops*

Problem	ProperLoops		ProperLoops*	
	number	time	number	time
2-6-2	439	0.42	68	0.10
2-6-3	559	0.73	76	0.14
2-7-1	657	0.56	115	0.15
2-7-2	1665	2.73	146	0.44
2-7-3	2790	14.63	162	1.15
2-8-1	2343	2.71	241	0.51
2-8-2	6389	16.90	304	1.93
2-9-1	8479	12.50	495	1.59
2-9-2	24532	220.10	622	13.83
2-10-1	32625	61.70	1005	4.78

Conclusion and Future Work

As a further refinement of the Lin-Zhao theorem, we have characterized a subclass proper loops of elementary loops. *RLF* loop formulas of proper loops allow us to disregard redundant loop formulas of loops and some elementary loops. As a result, a polynomial time algorithm is proposed to recognizing a proper loop and an algorithm is proposed to identifying all proper loops of a program using the structure of the positive body-head dependency graph. Experimental results show that, for programs whose dependency graphs consisting of sets of components with densely connected inside and sparsely connected outside, the algorithms could safely ignore a large number of loops and improve its performance.

We think the contributions open issues for future work:

- The notion of proper loops to normal logic programs could be extend to disjunctive logic programs, general logic programs, and propositional circumscription.
- We have shown that for certain programs identifying all proper loops is more efficient than identifying all elementary loops. Proper loops can be used in ASP solvers such as ASSAT, cmodels, and clasp directly to improve the efficiency.
- We have proven that, for programs whose dependency graphs consisting of sets of components that are densely connected inside and sparsely connected outside, after adding a small number of loop formulas of corresponding proper sets, loop formulas of loops assembled from different components could be ignored for answer set computation, which could benefit answer set computation.

Acknowledgments

We are grateful to Fangzhen Lin for many helpful and informative discussions. We would also like to thank Xiaoping Chen and his research group for their useful discussions. We are also grateful to Yongmei Liu for her useful suggestions. Jianmin Ji' research was partially supported by the Fundamental Research Funds for the Central Universities under grant WK0110000035, the National Natural Science Foundation of China under grant 61175057, as well as the USTC Key Direction Project and the USTC 985 Project. Hai Wan thanks Research Fund for the Doctoral Program of Higher Education of China (No. 20110171120041), Natural Science Foundation of Guangdong Province of China (No. S2012010009836), and Guangzhou Science and Technology Project (No. 2013J4100058) for the support of this research.

References

Chen, X.; Ji, J.; and Lin, F. 2013. Computing loops with at most one external support rule. *ACM Transactions on Computational Logic (TOCL)* 14(1):3–40.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2006. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence* 47(1-2):79–101.

Gebser, M., and Schaub, T. 2005. Loops: relevant or redundant? In *Proceedings of 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-05)*, 53–65.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 386–392.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming (ICLP-88)*, 1070–1080.

Giunchiglia, E.; Lierler, Y.; and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning* 36(4):345–377.

Lee, J., and Lifschitz, V. 2003. Loop formulas for disjunctive logic programs. In *Proceedings of the 19th International Conference on Logic Programming (ICLP-03)*, 451–465.

Lee, J., and Lin, F. 2006. Loop formulas for circumscription. *Artificial Intelligence* 170(2):160–185.

Lee, J., and Meng, Y. 2008. On loop formulas with variables. In *Proceedings of the 11th International Conference on Knowledge Representation and Reasoning (KR-08)*, 444–453.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, volume 5, 503–508.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM Transactions on Computational Logic* 7(2):261–268.

Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence* 157(1-2):115–137.

Linke, T., and Sarsakov, V. 2005. Suitable graphs for answer set programming. In *Logic for Programming, Artificial Intelligence, and Reasoning*, 154–168.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3):241–273.