Elementary Loops Revisited

Jianmin Ji\textsuperscript{a}, Hai Wan\textsuperscript{b}, Peng Xiao\textsuperscript{b}, Ziwei Huo\textsuperscript{b}, and Zhanhao Xiao\textsuperscript{c}
\textsuperscript{a}School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
jianmin@ustc.edu.cn
\textsuperscript{b}School of Software, Sun-yat Sen university, Guangzhou, China
wanhai@mail.sysu.edu.cn
\textsuperscript{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 \textit{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 \textit{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), emodels (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 Lifschitz 2003), general logic programs (Ferraris, Lee, and Lifschitz 2006), propositional circumscription (Lee and Liu 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 \textit{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 \textit{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:

\[ p_0 \leftarrow p_1, \ldots, p_k, \neg q_1, \ldots, \neg q_m, \]

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

Given a logic program \( P \), let \( \text{Atoms}(P) \) be the set of atoms in it. For a rule \( r \) of the form (1), let \( \text{head}(r) \) be its head \( p_0 \), \( \text{body}(r) \) the set \( \{ p_1, \ldots, p_k, \neg q_1, \ldots, \neg q_m \} \) of literals obtained from the body of the rule with \( \text{not} \) replaced by \( \neg \), \( \text{body}^+ (r) \) the set of atoms in its body, \( \{ p_1, \ldots, p_k \} \), and \( \text{body}^- (r) \) the set of atoms under \( \text{not} \) in its body, \( \{ q_1, \ldots, q_m \} \). With a slight abuse of the notion, we use \( \text{body}(r) \) also for the conjunction of the literals in it. Let \( R \) be a set of rules, we denote \( \text{head}(R) = \{ \text{head}(r) \mid r \in R \} \).
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) ⊆ S and body⁻(r) ∩ S = ∅. Furthermore, S satisfies a logic program P, if for each rule r in P, S satisfies body(r) implies head(r) ∈ 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 ∈ 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 ΓP(S).

Now a set S of atoms is an answer set of P iff S = ΓP(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 GP, 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 ∈ P such that p = head(r) and q ∈ 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 GP such that all the vertices in the path are in L, i.e., the L-induced subgraph of GP 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 P1:

\[ p \leftarrow \quad p \leftarrow r \quad q \leftarrow r \quad r \leftarrow p \quad r \leftarrow q. \]

Figure 1 shows the positive dependency graph of P1. P1 has six loops: \{p\}, \{r\}, \{q\}, \{p, r\}, \{r, q\}, and \{p, r, q\}.

![Figure 1: The positive dependency graph of Program P1](image)

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) ∈ L and L ∩ body⁺(r) = ∅. 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:

\[ \bigvee_{p \in L} p \supset \bigvee_{r \in R^-} body(r). \]

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):

\[ \bigwedge_{p \in L} p \supset \bigvee_{r \in R^-} body(r). \]

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 instance, 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:

\[ F_L \supset \bigvee_{r \in R^-} body(r). \]

**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) ∈ Y, body⁺(r) ∩ (X \ Y) ≠ ∅, and body⁻(r) ∩ Y = ∅. 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 P1 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:

\[ EC_P^0(X) = \emptyset, \]

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

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

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 \( \text{CLF}(L_1, P) \supseteq \text{CLF}(L_2, P) \).

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

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

5*: \( S \) satisfies \( \text{CLF}(L, P) \) for all unsubdued loops of \( P \).

Intuitively, a set \( Y \) of atoms is unbound in another set \( X \), iff there exists a rule \( r \) in \( P \) such that \( r \subseteq R^- (Y) \) and \( r \not\subseteq R^- (X) \). Let \( L \) be a loop of a program \( P \), all its nonempty proper subsets \( L' \) are unbound in \( L \) for all proper loops \( L' \subseteq R^-(L) \). 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. \( G^* := \text{the } L \setminus \{a\} \text{ induced subgraph of } G_P \);
3. \( SCC^* := \text{the set of SCCs of } G^* \);
4. for each \( C \in SCC^* \):
5. if \( R^- (C) \subseteq R^- (L) \) then return \( C \) else
6. \( G_C := \text{the } C \setminus \text{head}(R^- (C) \setminus R^- (L)) \text{ induced subgraph of } G^* \);
7. \( SCC_C := \text{the set of SCCs of } G_C \);
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 the 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 \( r \in R^- (C) \setminus R^- (L) \), \( \text{head}(r) \) must not be in the sub-loop \( L' \) not outbound in \( L \), \( \text{head}(r) \) must be in \( R^- (L') \), a contradiction with \( R^- (L') \subseteq R^- (L) \). So we can remove \( \text{head}(R^- (C) \setminus R^- (L)) \) 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 \( \text{ElementaryLoop}(L, P) \) returns either \( L \) or a set \( C \) of atoms such that \( C \) is a loop of \( P \), \( C \subseteq L \), and \( R^- (C) \subseteq R^- (L) \) in \( O(n^2) \), where \( n \) is the number of atoms in \( L \). \( \text{ElementaryLoop}(L, P) \) 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 \textit{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 \( \text{RLF}(L, P) \) to denote the implication:

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

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

\[
\bigwedge_{p \in L} p \supset \bot.
\]

Clearly, \( \text{RLF}(L, P) \) is a special case of \( \text{LF}(L, P) \). By \( \text{RLF}(L, P) \), 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 \( \text{RLF}(L_1, P) \supset \text{RLF}(L_2, P) \).

A loop \( L \) of a logic program \( P \) is called \textit{proper} if there does not exist another loop \( L' \) of \( P \) such that

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

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

8. \( S \) satisfies \( \text{RLF}(L, P) \) for all proper loops \( L \) of \( P \);
9. \( S \) satisfies \( \text{DLF}(L, P) \) for all proper loops \( L \) of \( P \).

When loop formulas are in the form of \( \text{RLF} \), 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 \( P_2 \) has three proper loops: \( \{q\} \), \( \{r, q\} \) and \( \{p, r, q\} \). \( \{p, r\} \) and \( \{p\} \) are not proper loops as \( R^- \{(p, r)\} = \{p \leftarrow r \} \) and \( R^- \{(p, r, q)\} = \{p \leftarrow r \} \). \( \{r\} \) 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
The set of proper loops \( R^- (L') \subset R^- (L) \). 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 \( L \subsetneq S \) and there does not exist another loop \( L' \subset S \) such that

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

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

- \( C \subset L \) and \( R^- (C) \subsetneq R^- (L) \), or
- \( R^- (C) \neq \emptyset \) and \( R^- (C) \subset R^- (L) \),

in \( O(n^2) \), where \( n \) is the number of atoms in \( S \). \( \text{ProperLoop}(L, P, S) \) returns \( L \) iff \( L \) is a proper loop of \( P \) under \( S \). Specially, \( \text{ProperLoop}(P, L, \text{Atoms}(P)) \) 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 \( \text{ProperLoop} \) 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 \( \text{head}(R^- (L)) \subseteq L' \), 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.

Intuitively, the function \( \text{ProperLoops}(P, S) \) 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 \( \text{ProperLoops}(P, S) \) 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 \( \text{ProperLoops}(P, \text{Atoms}(P)) \) 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 \( C_1 \)).

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 edges of the graph.

\( ^1 \)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.

\( ^2 \)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://ms.sysu.edu.cn/~ewh/properloop.html](http://ms.sysu.edu.cn/~ewh/properloop.html)
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 \( \text{Atoms}(P) \cup \text{body}(P) \), 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 \( a = \text{head}(r) \) and \( B = \text{body}(r) \),
- \((B, a)\) if there is a rule \( r \in P \) such that \( B = \text{body}(r) \) and \( a \in \text{body}^+(r) \).

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 \( C \cup \{ \text{body}(r) \mid \text{head}(r) \in C \text{ and } \text{body}^+(r) \cap C \neq \emptyset \} \), and there are two kinds of arcs in the graph:
- \((a, B)\) if there is a rule \( r \in P \) such that \( a = \text{head}(r) \), \( B = \text{body}(r) \), \( a \in C \), and \( \text{body}^+(r) \cap C \neq \emptyset \),
- \((B, a)\) if there is a rule \( r \in P \) such that \( B = \text{body}(r) \), \( a \in \text{body}^+(r) \), \( a \in C \), and \( \text{head}(r) \in C \).

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.

Intuitively, if we can “remove” the vertex \( \text{body}(r_1) \) or \( \text{body}(r_2) \) 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 \( \text{body}(r_1) \) or \( \text{body}(r_2) \), the number of loops is reduced to \( 2^m + 2^n - 2 \).

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 \( r \in R \), \( \text{body}^+(r) \cap C \neq \emptyset \), and there does not exist another set \( C' \subseteq S \), for each \( r \in R \), \( \text{body}^+(r) \cap C' \neq \emptyset \) 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 \( \text{pset}(P, S, R) \) 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 \( r \in R \), \( \text{body}^+(r) \cap C \neq \emptyset \).

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

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

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

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

\[ \text{body}(r_1) \subset \text{body}(r_2) \]

\[ \text{body}(r_3) \subset \text{body}(r_4) \]

\[ \text{body}(r_5) \subset \text{body}(r_6) \]
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 $R = \text{edge}(S_1, S_2) \cup \text{edge}(S_2, S_1)$. 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 $\text{MinimalSeparator}(P, L)$ 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 vertices in $\text{Atoms}(P)$;
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 vertices with an edge valued with 1 between both vertices;
5. $\text{Cut} :=$ a minimum cut of $G_3$ computed by the Stoer-Wagner algorithm;
6. $R :=$ the set of corresponding rules for $\text{Cut}$;
7. return $R$

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

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

1. $\text{Loops} := \emptyset$;
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 $\text{Loops}$;
4. $\text{Pset}_1 := \emptyset$ and $\text{Pset}_2 := \emptyset$;
5. for each nonempty subset $R_1 \in \text{edge}(S_2, S_1)$:
6. append new loops in ProperSets$(P, S_1, R_1)$ to $\text{Pset}_1$;
7. for each nonempty subset $R_2 \in \text{edge}(S_1, S_2)$:
8. append new loops in ProperSets$(P, S_2, R_2)$ to $\text{Pset}_2$;
9. for each pair $(C_1, C_2)$ in $\text{Pset}_1 \times \text{Pset}_2$:
10. append the set $C_1 \cup C_2$ to $\text{Loops}$;
11. return $\text{Loops}$

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 $S_2$, $|S_1| = m$, $|\text{edge}(S_2, S_1)| = k_1$, for each nonempty subset $R_1 \subseteq \text{edge}(S_2, S_1)$, $|\text{ProperSets}(P, S_1, R_1)| = C_1$, $|\text{edge}(S_1, S_2)| = k_2$, and for each nonempty subset $R_2 \subseteq \text{edge}(S_1, S_2)$, $|\text{ProperSets}(P, S_2, R_2)| = C_2$, 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 $2^m + 2^{n-m} + k_1k_2C_1C_2$. 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 $2^n$.

Table 2 contains the numbers of checked loops and running times of ProperLoops$(P, \text{Atoms}(P))$ and ProperLoops$^*(P, \text{Atoms}(P))$ for corresponding HC programs. As can be seen, both numbers and running times are less for the function ProperLoops$^*(P, \text{Atoms}(P))$. Note that, for some programs, the number of considered loops for ProperLoops$^*(P, \text{Atoms}(P))$ 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$^*$

<table>
<thead>
<tr>
<th>Problem</th>
<th>ProperLoops</th>
<th>ProperLoops$^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number</td>
<td>time</td>
</tr>
<tr>
<td>2-6-2</td>
<td>439</td>
<td>0.42</td>
</tr>
<tr>
<td>2-6-3</td>
<td>559</td>
<td>0.76</td>
</tr>
<tr>
<td>2-7-1</td>
<td>657</td>
<td>0.56</td>
</tr>
<tr>
<td>2-7-2</td>
<td>1665</td>
<td>2.73</td>
</tr>
<tr>
<td>2-7-3</td>
<td>2790</td>
<td>14.63</td>
</tr>
<tr>
<td>2-8-1</td>
<td>2343</td>
<td>2.71</td>
</tr>
<tr>
<td>2-8-2</td>
<td>6389</td>
<td>16.90</td>
</tr>
<tr>
<td>2-9-1</td>
<td>8479</td>
<td>12.50</td>
</tr>
<tr>
<td>2-9-2</td>
<td>24532</td>
<td>220.10</td>
</tr>
<tr>
<td>2-10-1</td>
<td>32625</td>
<td>61.70</td>
</tr>
</tbody>
</table>

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. Jianmin Ji’s research was partially supported by the Fundamental Research Funds for the Central Universities under grant WK0110000035, the National Natural Science Foundation of China under grand 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. S201201009836), and Guangzhou Science and Technology Project (No. 2013J4100058) for the support of this research.

References


