A Counting Algorithm for a Cyclic Binary Query

RAMSEY W. HADDAD*

Computer Science Department Stanford University

AND

JEFFREY F. NAUGHTON†

Computer Science Department Princeton University

Received September 27, 1988; revised January 9, 1990

In this paper we consider selection queries on a generalization of the Datalog program known as the "same generation." This program has received a great deal of attention in the literature because selection queries on this simple, binary program have no monadic equivalent. As such, the program encapsulates many of the difficulties that arise in more general recursive query processing. In general, counting methods perform well on such queries. However, counting methods fail in the presence of cycles in the database. We present an algorithm in the spirit of counting methods that correctly deals with cyclic data and has the same asymptotic running time as counting methods. The algorithm, which is based on reducing a query on a database to a question about intersections of semilinear sets, works by using efficient methods to construct the appropriate semilinear sets from the database and query constant. © 1991 Academic Press, Inc.

1. INTRODUCTION

In this paper we are concerned with answering single-selection queries on Datalog programs consisting of a single linear recursive rule and a nonrecursive rule. (Datalog is the subset of pure Prolog containing no function symbols—that is, all terms are either variables or constants.) The specific recursion we consider is expressed in Datalog as

\[ t(X, Y) := b(X, Y). \]

\[ t(X, Y) := a(X, W), t(W, Z), c(Z, Y). \]

and the query \( t(x_0, Y)? \).

* The work of this author was supported by NSF grants DCR 83-08109 and CCR 86-10181. Author's current address: DEC Western Research Lab, Palo Alto, CA.

† The work of this author was supported in part by NSF grant IRI-8909795. Author's current address: Department of Computer Sciences, University of Wisconsin, Madison, WI.
The motivation for studying this recursion is that, informally, it is the simplest Datalog recursion for which it is impossible to use a selection to reduce the arity of the recursively defined predicate. If a selection on a given recursion can be used to reduce the arity of the recursion, then simple, efficient evaluation algorithms result (see e.g., [NRSU89]). For example, a "column = constant" selection on either column of the transitive closure can be converted to a monadic recursion; evaluating the resulting monadic recursion corresponds to performing a depth-first search rooted at the query constant, and hence can be done in linear time.

A proof that the query \( t(x_0, Y) \) as defined by the preceding Datalog program has no monadic equivalent appears in [BKBR87]. Because there is no monadic equivalent, finding an efficient evaluation algorithm for the recursion is not straightforward. In this paper we present an algorithm for this query that is asymptotically the best known to date. The algorithm and its development give insight into the difficulties involved in devising efficient evaluation algorithms for queries for which the selection cannot be used to reduce the arity.

This query is an abstraction of the well-known "same generation" program:

\[
\text{sg}(X, Y) :- X = Y, \text{person}(X).
\]

\[
\text{sg}(X, Y) :- \text{par}(X, W), \text{sg}(W, Z), \text{par}(Y, Z).
\]

In words, every person \( X \) is of the same generation as his or herself. Also, if a person \( X \) has a parent \( W \), and a person \( Y \) has a parent \( Z \), and \( W \) and \( Z \) are of the same generation, then so are \( X \) and \( Y \). Suppose we wish to know all the people at the same generation as a given person, say \( \text{tom} \). This gives rise to the Datalog query \( \text{sg(tom, Y)} \)?

The recursion and query we consider in this paper can be rephrased as a problem in finding efficient graph algorithms. The database can be interpreted as a graph with three kinds of directed edges, with labels \( a \), \( b \), and \( c \), respectively. There is an edge labeled \( a(b, c) \) from \( v_1 \) to \( v_2 \) if and only if the tuple \((v_1, v_2)\) appears in relation \( a(b, c) \). The query \( t(x_0, Y) \)? is then asking for all nodes \( Y \) such that, for some \( i \), there is a path from \( x_0 \) to \( Y \) consisting of \( i \) edges labeled \( a \), then a \( b \)-edge, and finally \( i \) more \( c \)-edges.

Algorithms proposed in the database literature to date to answer this query can be divided into two broad categories.

Algorithms in the first category, which we call Magic Set algorithms, first determine the relevant portion of the database being queried, and then use seminaive bottom-up evaluation to evaluate the original recursion on this relevant subset of the original database. The Magic Sets algorithm was first proposed by Bancilhon et al. [BMSU86]. Various extensions and modifications to that original algorithm have been proposed by Sacca and Zaniolo [SZ86b] and by Beeri and Ramakrishnan [BR87].

Algorithms in the second category, which we call the counting algorithms, reduce the arity of the recursion by storing information about how each tuple in the intermediate relations was derived. The name "counting methods" was first applied to
two methods appearing in Bancilhon et al. [BMSU86]. Extensions and modifications of those algorithms were later explored [SZ86a, BR87].

Although it is quite different from the methods usually referred to as counting methods, we consider the algorithm proposed in an early paper by Henschen and Naqvi [HN84] to be in the spirit of the counting methods, as it implicitly stores information about how tuples in the intermediate relations were derived. Similar comments apply to algorithms presented by Grahne et al. [GSSS87] and by Han and Henschen [HH87].

Comparing counting and noncounting algorithms is difficult, as noncounting algorithms use only relational operations, whereas counting algorithms use other operations (typically arithmetic) as well. However, by assuming that arithmetic operations have the same cost as tuple-element comparisons, Bancilhon and Ramakrishnan [BR88] have found that on a variety of databases the counting method outperforms magic sets by a factor of 10. In another study, Marchetti-Spaccamela et al. [MSPS87] report that, on the query \( t(x_0, Y) \) from above, Magic Sets is \( \Theta(e^2) \) while counting is \( \Theta(ne) \), where \( n \) and \( e \) are the number of nodes and edges in the graph represented by the database.

The difficulty with the counting methods is that they fail if the database is cyclic. (In our example, the database is cyclic if there is a path from some node \( v \) back to \( v \) consisting either of \( a \)-edges or of \( c \)-edges.) Intuitively, the reason is that, in a cyclic database, some tuples in the intermediate relations can be derived in infinitely many ways. Since the counting methods store an intermediate relation tuple once for each different derivation, they will never terminate.

An initial attempt at a counting algorithm that deals with cyclic data appeared in Henschen and Naqvi [HN84], but the termination condition given in that paper is premature. A proof that the termination condition is premature is given in Briggs [Bri86]. Han and Henschen [HH87] give a counting algorithm “level cycle merging” that in the worst case requires exponential preprocessing of the database on each update and is \( O(e^2) \) to answer each query. Other authors have noted that every answer tuple must have at least one derivation of length \( \leq n^2 \). Marchetti-Spaccamela et al. [MSPS87] use this fact to produce an \( O(ne^2) \) algorithm, while Grahne et al. [GSSS87] use this fact to produce an algorithm that is \( O(n^3) \). Sacca and Zaniolo [SZ87] propose a method that runs a counting algorithm until a cycle is detected, then switches over to Magic Sets. This algorithm is \( O(ne) \) on acyclic data, and \( O(e^2) \) on cyclic data.

The method we present in this paper is \( O(ne) \) and correctly handles cyclic data. The basic idea is to reduce the problem to one of detecting intersections of semilinear sets, and then to develop efficient ways of computing the relevant semilinear sets and their intersections from the database and the query constant.

2. Overview

The query \( t(x_0, Y) \) asks for all \( Y \) such that there is a path that starts at \( x_0 \) and consists of \( i \) consecutive \( a \)-edges, followed by one \( b \)-edge, followed by \( i \) consecutive
c-edges, for some \( i \geq 0 \). If we consider \( a \)-edges to have weight 1, \( b \)-edges to have weight 0, and \( c \)-edges to have weight \(-1\), then the query can be viewed as asking for all \( Y \) such that there is an \( a^*bc^* \) path from \( x_0 \) to \( Y \) of weight 0.

All nodes reachable from \( x_0 \) by paths of the form \( a'bc' \) such that there are either no cycles of \( a \)-edges or no cycles of \( c \)-edges can be found in \( O(ne) \) time by running the basic counting algorithm from [BMSU86] for \( n' \leq n \) steps, where \( n' \) is the longest acyclic path in the relevant portion of \( a \) or \( c \). Hence, in the sequel we assume that all such answers have been found, and we focus our attention solely on finding all nodes reachable from \( x_0 \) by \( a'bc' \) paths that contain both cycles of \( a \)-edges and cycles of \( c \)-edges.

2.1. Distance Sets

A key notion in the remainder of this paper is that of a distance set.

**Definition 2.1.** For a directed graph \( G \) and two vertices \( x \) and \( y \) in \( V \), the distance set \( D(x, y) = \{ w : \text{there is a path in } G \text{ from } x \text{ to } y \text{ of weight } w \} \). This includes nonsimple paths. Also, define \( D_a^*(x, y) = \{ w : \text{there is a path of the form } a^* \text{ from } x \text{ to } y \text{ of weight } w \} \) and \( D_{bc}^*(x, y) = \{ |w| : \text{there is a path of the form } bc^* \text{ from } x \text{ to } y \text{ of weight } w \} \).

Each distance set will be split into two distance sets: The distance sets \( AD(x, y) \)—the acyclic distance sets—represent paths that have not yet encountered a cycle and the distance sets \( CD(x, y) \)—the cyclic distance sets—represent paths that have encountered cycles.

**Example 2.2.** For the graph in Fig. 1, we have

\[
CD(x, y) = \{ 7 + 4i : i \geq 0 \} \cup \{ 9 + 4i : i \geq 0 \} \cup \{ 7 + 2j + 4k : j, k \geq 0 \}
\]

![Fig. 1. A simple graph.](image-url)
and

\[ \text{AD}(x, y) = \emptyset \]

**Definition 2.3.** Let \( S \) be a set of integers and \( n \) be a positive integer. Then 
\( S \mod n = \{ s \mod n : s \in S \} \). For two sets of integers \( S \) and \( T \), \( S + T = \{ s + t : s \in S, t \in T \} \). Define \( S - T \) similarly. Define \( T \setminus S = \{ s : s \in T, s \notin S \} \).

In the following we use the standard notation \( \mathcal{N}_0 \) to refer to the set of positive integers with zero added, and \( \mathcal{N}_1 \) to refer to the positive integers (without zero.)

**Definition 2.4.** Let \( S \) be a set of integers. If there exists an integer \( p > 0 \) and a set of integers \( C \), with all \( c \in C \) satisfying \( 0 \leq c < p \), such that 
\( S = \{ c + \lambda p : c \in C, \lambda \in \mathcal{N}_0 \} \), then we call \( S \) periodic. We use the notation \( L(C; p) \) to refer to a specific representation—choice of \( C \) and \( p \)—of \( S \). Any periodic set \( S \) has numerous representations. Note that the union of two periodic sets is itself a periodic set.

**Example 2.5.** The set \( S = \{ 2, 5, 8, 11, \ldots \} \) is periodic with \( C = \{ 2 \} \) and \( p = 3 \). Another representation of the same set is \( C = \{ 2, 5 \} \) and \( p = 6 \). The set \( S' = \{ 5, 7, 9, 11, \ldots \} \) is not periodic, since it cannot be represented in form \( L(C; p) \) with \( c < p \) for all \( c \in C \).

The following lemma shows how to add two periodic sets.

**Lemma 2.6.** Let \( L(C_1; p_1) \) and \( L(C_2; p_2) \) be periodic sets. Then \( L(C_1; p_1) + L(C_2; p_2) = L(C_1 + C_2; (p_1, p_2)) \).

**Proof.** First, consider an arbitrary element \( s \in L(C_1 + C_2; (p_1, p_2)) \). By definition of \( L(C_1 + C_2; (p_1, p_2)) \), we must have constants \( c_1 \in C_1 \) and \( c_2 \in C_2 \) and integers \( \lambda_1 \) and \( \lambda_2 \) such that

\[
\begin{align*}
  s &= c_1 + \lambda_1 p_1 + \lambda_2 p_2 \\
  &= (c_1 + \lambda_1 p_1) + (c_2 + \lambda_2 p_2) \\
  &= s_1 + s_2,
\end{align*}
\]

where \( s_1 \in L(C_1; p_1) \) and \( s_2 \in L(C_2; p_2) \), so \( L(C_1; p_1) + L(C_2; p_2) \subseteq L(C_1 + C_2; (p_1, p_2)) \).

Next consider arbitrary elements \( s_1 \in L(C_1; p_1) \) and \( s_2 \in L(C_2; p_2) \). Then

\[
\begin{align*}
  s_1 + s_2 &= (c_1 + \lambda_1 p_1) + (c_2 + \lambda_2 p_2) \\
  &= c_1 + c_2 + \lambda_1 p_1 + \lambda_2 p_2 \\
  &= s,
\end{align*}
\]
where \( s \in L(C_1 + C_2; (p_1, p_2)) \). So \( L(C_1 + C_2; (p_1, p_2)) \subseteq L(C_1; p_1) + L(C_2; p_2) \), which completes the proof.

As computing distance sets is prohibitively expensive we instead compute simplified distance sets, which are periodic sets that “approximate” the actual distance set. While we say more about them later, for now, we just state that the cyclic simplified distance sets \( CS(x, y) \) have the following properties:

**Property 1.** \( CD(x, y) \subseteq CS(x, y) \).

**Property 2.** \( CS(x, y) \setminus CD(x, y) \) is a finite set.

**Property 3.** \( CS(x, y) \) is periodic.

**Example 2.7.** As shown below, the simplified cyclic distance set for the cyclic distance set \( CD(x, y) \) from Example 2.2 is expressible with \( C = \{1, 3\} \) and \( p = 4 \).

### 2.2. Tests

The obvious test for the question, “Is there a path of the form \( a'bc'd \) from a node \( x \) to node \( y \)” is:

**Test 1.** \( t(x, y) \iff \) There exists a node \( z \) such that \( CD_{a'}(x, z) \cap CD_{bc'}(z, y) \neq \emptyset \).

If \( CD_{a'}(x, z) \cap CD_{bc'}(z, y) \neq \emptyset \), then there is a weight \( w_0 \in CD_{a'}(x, z) \cap CD_{bc'}(z, y) \). By definition of \( CD \), it follows that there is a path of the form \( a^* \) with weight \( w_0 \) from \( x \) to \( z \) that has at least one cycle on it, say of weight \( w_1 \). Similarly, there is a path of the form \( bc^* \) with weight \( w_0 \) from \( z \) to \( y \) that has at least one cycle of \( c^* \)-edges in it, say of length \( w_2 \). Clearly for any integer \( i \geq 0 \), \( w_0 + iw_1w_2 \in CD_{a'}(x, z) \cap CD_{bc'}(z, y) \). This leads us to an equivalent new test:

**Test 2.** \( t(x, y) \iff \) There exists a node \( z \) such that \( CD_{a'}(x, z) \) and \( CD_{bc'}(z, y) \) intersect infinitely often.

If \( CD_{a'}(x, z) \) and \( CD_{bc'}(z, y) \) intersect infinitely often, then by Property 1, \( CS_{a'}(x, z) \) and \( CS_{bc'}(z, y) \) intersect infinitely often. Similarly, if \( CS_{a'}(x, z) \) and \( CS_{bc'}(z, y) \) intersect infinitely often, then Property 2 implies that \( CD_{a'}(x, z) \) and \( CD_{bc'}(z, y) \) intersect infinitely often. Hence, another equivalent test is:

**Test 3.** \( t(x, y) \iff \) There exists a node \( z \) such that \( CS_{a'}(x, z) \) and \( CS_{bc'}(z, y) \) intersect infinitely often.

Trivially, if \( CS_{a'}(x, z) \) and \( CS_{bc'}(z, y) \) intersect infinitely often, then they intersect at least once. More interestingly, Property 3 implies that if \( CS_{a'}(x, z) \) and \( CS_{bc'}(x, y) \) intersect at least once then they intersect infinitely often. Thus, the next test is also equivalent:

**Test 4.** \( t(x, y) \iff \) There exists a node \( z \) such that \( CS_{a'}(x, z) \cap CS_{bc'}(z, y) \neq \emptyset \).

This is clearly equivalent to the test:
1. Find the set of nodes $V_a$ that are reachable from $x_0$ by paths of $a$-edges.
2. Compute the distance sets $AS_{a^*}(x_0, x)$ and $CS_{a^*}(x_0, x)$ of the nodes $x \in V_a$.
3. Let $V'_a$ be those nodes of $V_a$ that are reachable from $x_0$ by paths with cycles, that is $V'_a = \{ x \in V_a : CS_{a^*}(x_0, x) \neq \emptyset \}$.
4. Find the set of nodes $V_b$ that are reachable from $V'_a$ by a single $b$-edge.
5. Find the set of nodes $V_c$ that are reachable from $V_b$ by paths of $c$-edges.
6. Compute the simplified distance sets $AS_{a^*b^*c^*}(x_0, y)$ and $CS_{a^*b^*c^*}(x_0, y)$ for each of the nodes $y \in V_c$.
7. For each $y \in V_c$, check if $0 \in CS_{a^*b^*c^*}(x_0, y)$. If so, $(x_0, y)$ is in $t$.

**Fig. 2.** Overview of algorithm.

**Test 5.** $t(x, y) \iff$ There exists a node $z$ such that $0 \in CS_{a^*}(z, z) - CS_{b^*}(z, y)$.

Finally, by using the definition: $CS_{a^*b^*c^*}(x_0, y) = \bigcup_z CS_{a^*}(z, z) - CS_{b^*}(z, y)$ we get our final version of the test:

**Test 6.** $t(x, y) \iff 0 \in CS_{a^*b^*c^*}(x, y)$.

This last test is the test that forms the basis of the algorithm, which we can now sketch in Fig. 2. Note that $AS$ is a simplified version of $AD$. We still need to: (1) define the cyclic simplified distance sets rigorously and show that they have the three claimed properties and (2) explain how to compute them efficiently.

### 3. SEMILINEAR SETS AND SIMPLIFIED DISTANCE SETS

While these distance sets mentioned in Section 2 can be infinite sets, their structure allows them to be easily represented and manipulated.

**Definition 3.1.** A set $D$ is **linear** if it is of the form

$$D = \left\{ c + \sum_{p \in P} \lambda_p p : \lambda_1, \lambda_2, \ldots \in \mathcal{N}_0 \right\}$$

for some $c \in \mathcal{N}_0$ and some finite set $P \subset \mathcal{N}_1$. A set $D$ is **semilinear** if it can be expressed as a finite union of linear sets.

While all our sets are semilinear sets and can be represented as unions of linear sets, we actually represent them as unions of more complicated sets.

- **Periodic Sets:** $L(C; p) = \{ c + \lambda p : c \in C, \lambda \in \mathcal{N}_0 \}$ such that $C$ is finite, $p > 0$, and $0 \leq c < p$ for all $c \in C$.
- **Aperiodic Sets:** Finite set $C$. Sometimes written as either $L(C; 0)$ or $L(C; \emptyset)$.
- **Multiperiodic Sets:** $L(C; P) = \{ c + \sum_{p \in P} \lambda_p p : c \in C \text{ and } \lambda_1, \lambda_2, \ldots \in \mathcal{N}_0 \}$ such that $C$ and $P$ are finite, $P \neq \emptyset$, and $p > 0$ for all $p \in P$. 

Our algorithm only needs periodic and aperiodic sets. However, multiperiodic sets are needed for the discussion of the correctness of our algorithm.

The following lemma shows why we are interested in semilinear sets.

**Lemma 3.2.** For a finite directed graph $G$, the distance sets $D(x, y)$ for $x, y \in V$ are semilinear sets.

**Proof.** Parikh [Par66] proved that $\{i \mid 0^i \in L\}$ is semilinear for context-free languages $L$. Hence, it is trivially true for regular languages $L$ as well.

The strongly connected component of a directed graph $G$ containing $x$ and $y$ can be viewed as a finite automaton by treating $x$ as a start node, $y$ as a terminal node, and giving each edge the label $0$. This automaton defines a regular language $L$ such that $\{i \mid 0^i \in L\}$ is just $D(x, y)$. So distance sets are semilinear.

Any semilinear set can be represented as the union of a single aperiodic set $L(C_0; \emptyset)$ with a finite union of multiperiodic sets $\bigcup_{i=1}^n L(C_i; P_i)$. The cyclic distance sets $CD(x, y)$ have the useful property that each can be represented as a finite union of multiperiodic sets. This can be seen as follows:

Suppose that $CD(x, y) = L(C_0; \emptyset) \cup \bigcup_{i=1}^n L(C_i; P_i)$. If $C_0 = \emptyset$ then we are done. Hence assume that we have some $c \in C_0$. Since $c \in CD(x, y)$, then $c$ is on some path from $x$ to $y$ that has a cycle, say of weight $w$. By removing $c$ from the set $L(C_0; \emptyset)$ and adding a new set $L(\{c\}; \{w\})$, we have a new equivalent representation for the set $CD(x, y)$ that has a smaller $C_0$. We can repeat this until $C_0 = \emptyset$.

**Example 3.3.** The cyclic distance set $CD(x, y)$ for the graph in Fig. 1 is a semilinear set. It is expressible as a union of multiperiodic sets $\bigcup_{i=1}^n L(C_i; P_i)$, where $C_1 = \{7, 9\}$ and $P_1 = \{4\}$, and $C_2 = \{7\}$ and $P_2 = \{2, 4\}$.

If a semilinear set $S$ can be represented as the union of a finite number of multiperiodic sets $\bigcup_{i=1}^n L(C_i; P_i)$, then it can be closely approximated by a periodic set. The phrase "closely approximated" is made precise by the following lemma:

**Lemma 3.4.** Let $S$ be representable as a finite union of multiperiodic sets $\bigcup_{i=1}^n L(C_i; P_i)$. Then there is a periodic set $T$ such that $S \subseteq T$ and such that $T \setminus S$ is a finite set.

**Proof.** The proof proceeds by defining two functions, $B(\cdot)$ and $M(\cdot)$, such that for any set $S$ of the specified form, $T = M(B(S))$.

First we define $B(\cdot)$ for a single multiperiodic set $L(C; P)$. Since $P \neq \emptyset$ and $p > 0$ for all $p \in P$, then $g = \gcd(P)$ is well defined. We define the function $B(L(C; P))$ as $B(L(C; P)) = L(C \mod g; g)$.

The function $B$ maps any multiperiodic set $S = L(C; P)$ into a periodic set. Clearly
$B(S) \setminus S$ is a finite set and $S \subseteq B(S)$. We extend this definition to apply to unions of multiperiodic sets $L(C; P)$ by setting

$$B\left( \bigcup_{i=1}^{m} L(C_i; P_i) \right) \overset{\triangle}{=} \bigcup_{i=1}^{m} B(L(C_i; P_i)).$$

It is still true that $B(S) \setminus S$ is a finite set and $S \subseteq B(S)$.

Since a union of periodic sets is itself periodic, then $B(S)$ is a periodic set. However, the function $B()$ returns it as a union of periodic sets instead of as a single periodic set. To find a representation for $B(S)$ as a single periodic set, we need the function $M()$. Next, let $S' = \bigcup_{i=1}^{m} L(C_i; P_i)$, and let $l = \text{lcm}(p_1, \ldots, p_m)$. Then we define the set function $M(\bigcup_{i=1}^{m} L(C_i; P_i))$ as

$$M\left( \bigcup_{i=1}^{m} L(C_i; P_i) \right) \overset{\triangle}{=} L\left( \bigcup_{i=1}^{m} (L(C_i; P_i) \mod l); l \right).$$

As mentioned above, all we did is find a new representation for the set $S'$ and hence $S' = M(S')$. Clearly $T = M(B(S))$ is periodic. Furthermore, $S \subseteq B(S) = M(B(S)) = T$; hence, $S \subseteq T$. Lastly, since $B(S) \setminus S$ is finite, then so is $T \setminus S$.

**Example 3.5.** We now compute the set $T = M(B(\text{CD}(x, y)))$ for the cyclic reachability set $\text{CD}(x, y) = L(\{7, 9\}, \{4\}) \cup L(\{7\}, \{2, 4\})$ from the graph of Fig. 1.

$$B(L(\{7, 9\}, \{4\})) = L(\{1, 3\}, 4)$$

$$B(L(\{7\}, \{2, 4\})) = L(\{1\}, 2)$$

$$B(L(\{7, 9\}, \{4\}) \cup L(\{7\}, \{2, 4\}) = L(\{1, 3\}, 4) \cup L(\{1\}, 2)$$

$$M(L(\{1, 3\}, 4) \cup L(\{1\}, 2)) = L(\{1, 3\}, 4).$$

Enumerating the original set $\text{CD}(x, y)$ we get $\{7, 9, 11, \ldots\}$ whereas $M(B(\text{CD}(x, y))) = \{1, 3, 5, \ldots\}$. Clearly $\text{CD}(x, y) \subseteq M(B(\text{CD}(x, y)))$, and $\text{CD}(x, y) \setminus M(B(\text{CD}(x, y)))$, which is just $\{1, 3, 5\}$, is a finite set.

The following lemmas show that for a set $S$ in the specified form, the approximating periodic set $T$ is unique.

**Lemma 3.6.** Let $T_1 = L(C_1; p_1)$ and $T_2 = L(C_2; p_2)$ be periodic sets. Either $T_1 = T_2$ or the set $U = (T_1 \cup T_2) \setminus (T_1 \cap T_2)$ is an infinite set.

**Proof.** If $U \neq \emptyset$ then there is some integer $t \in U$. We can assume without loss of generality that $t \in T_1$ and that $t \notin T_2$. It follows that $t + kp_1, p_2 \in T_1$ and $t + kp_1, p_2 \notin T_2$ for every $k \geq 0$. Hence, $t + kp_1, p_2 \in U$ for every $k \geq 0$. This implies that $U$ is infinite.
**Lemma 3.7.** Let $S = \bigcup_{i=1}^m L(C_i; P_i)$ be a finite union of multiperiodic sets. Also, let there be two periodic sets $T_1$ and $T_2$ such that $S \subseteq T_1$, $S \subseteq T_2$, $T_1 \setminus S$ is finite and $T_2 \setminus S$ is finite. Then, $T_1 = T_2$ (they may, though, have different representations).

**Proof.** Let $T_1 = L(C_1; p_1)$ and $T_2 = L(C_2; p_2)$. Let $U = (T_1 \cup T_2) \setminus (T_1 \cap T_2)$. Since $T_1 \cap T_2 \subseteq S$, then $U = (T_1 \cup T_2) \setminus (T_1 \cap T_2) \subseteq (T_1 \setminus S) \cup (T_2 \setminus S)$. Since $T_1 \setminus S$ and $T_2 \setminus S$ are finite, then $U$ is finite. By Lemma 3.6, $U = \emptyset$ and $T_1 = T_2$.  

We define the function $R(\ )$ (for “reduce”) as follows.

**Definition 3.8.** Let $S$ be a semilinear set. Then we define $R(S)$ by the equation $R(S) = M(B(S))$.

The function $R(\ )$ maps a finite union of multiperiodic sets $L(C; P)$ to the associated approximating periodic set. For a cyclic distance set $CD(x, y)$ we take $CS(x, y) = R(CD(x, y))$ to be its corresponding simplified set. In Section 4 we need the following properties of the operator $R(\ )$.

**Lemma 3.9.** Let $D_1$ be a multiperiodic set and $D_2$ be another set. Then

$$R(D_1 \cup D_2) = R(R(D_1) \cup D_2).$$

**Proof.** It is clear that $R(D_1 \cup D_2) \subseteq R(R(D_1) \cup D_2)$ and that $R(R(D_1) \cup D_2) \setminus R(D_1 \cup D_2)$ is finite. Hence, by Lemma 3.6 these two periodic sets are actually equal.

**Lemma 3.10.** Let $D_1$ be a multiperiodic set and let $D_2$ be either a finite set or a finite union of multiperiodic sets. Then

$$R(D_1 + D_2) = R(R(D_1) + D_2).$$

**Proof.** Clearly $R(D_1 + D_2) \subseteq R(R(D_1) + D_2)$. If $D_2$ is finite then $R(R(D_1) + D_2) \setminus R(D_1 + D_2)$ is finite. And hence, by Lemma 3.6 these two periodic sets are actually equal.

If $D_2$ is a finite union of multiperiodic sets, we can see that $B(D_1 + D_2) = B(B(D_1) + D_2)$ by comparing the last lines of the following two equations. We can replace $B(\ )$ with $R(\ )$, by inserting $M(\ )$'s because all $M(\ )$ does is re-represent the set and Lemma 3.7 tells us that the result is representation independent.

$$B(D_1 + D_2) = B\left( \bigcup_i L(C_i; P_i) + \bigcup_j L(C_j; P_j') \right)$$

$$= B\left( \bigcup_i \bigcup_j (L(C_i; P_i) + L(C_j; P_j')) \right)$$

$$= \bigcup_i \bigcup_j B(L(C_i; P_i) + L(C_j; P_j'))$$

$$= \bigcup_i \bigcup_j L(C_i + C_j \mod \gcd(P_i \cup P_j'; \gcd(P_i \cup P_j'))$$
A CYCLIC BINARY QUERY

$$B(B(D_1) + D_2) = B\left( B\left( \bigcup_i L(C_i; P_i) \right) + \bigcup_j L(C'_j; P'_j) \right)$$

$$= B\left( \bigcup_i B(L(C_i; P_i)) + \bigcup_j L(C'_j; P'_j) \right)$$

$$= B\left( \bigcup_i L(C_i \mod \gcd(P_i); \gcd(P_i)) + \bigcup_j L(C'_j; P'_j) \right)$$

$$= B\left( \bigcup_i \bigcup_j L(C_i \mod \gcd(P_i); \gcd(P_i)) + L(C'_j; P'_j) \right)$$

$$= \bigcup_i \bigcup_j B(L(C_i \mod \gcd(P_i); \gcd(P_i)) + L(C'_j; P'_j))$$

$$= \bigcup_i \bigcup_j L((C_i \mod \gcd(P_i)) + C'_j \mod \gcd(\{\gcd(P_i)\} \cup P'_j); \gcd(\{\gcd(P_i)\} \cup P'_j))$$

$$= \bigcup_i \bigcup_j L(C_i + C'_j \mod \gcd(P_i \cup P'_j); \gcd(P_i \cup P'_j)).$$

4. Computing Distance Sets

Steps 1, 3, 4, 5, and 7 of the algorithm in Fig. 2 are straightforward. This section describes how to perform steps 2 and 6. We begin by describing how to perform step 2; the modifications required to perform step 6 are described at the end of this section. Recall that in step 2 we are interested only in the subgraph induced by the nodes in $V_a$ and their associated $a$-edges; in particular “path” means “path of $a$-edges.” Figure 3 shows the steps in step 2 used to compute the simplified distance sets $A_{S_a^a}(x_0, y)$ and $C_{S_a^a}(x_0, y)$.

We now describe the steps in Fig. 3 in more detail.

2. Compute the distance sets $A_{S_a^a}(x_0, x)$ and $C_{S_a^a}(x_0, x)$ of the nodes $x \in V_a$.
   (a) Create a fake node $x_0$ with one outgoing $a$-edge of weight 0 to $x_0$.
   (b) Perform a depth first search from $x_0$ along paths of the form $a^*$ to determine those nodes reachable from $x_0$ and to determine the maximal strongly connected components of the graph induced by those nodes.
   (c) Next compute the period of each maximal strongly connected component and collapse the nodes of a component into equivalence classes.
   (d) Simplify the inter-components edges. This produces a DAG, where each node of the DAG represents a connected component and the edges of the DAG are the simplified inter-component edges.
   (e) Working from the sources to the sinks of the DAG, compute the simplified distance set of each component representative.

Fig. 3. More details of step 2.
4.1. **Fake Nodes**

The fake node allows us to describe the computation of $AS_a^*(x_0, Y)$ and $GS_a^*(x_0, Y)$ in a manner that is equally applicable to computing $AS_{a*bc}^*(x_0, Y)$ and $CS_{a*bc}^*(x_0, Y)$. The fake node $x_0'$ has a single $a$-edge of weight 0 to the node $x_0$, and is initialized by setting $CS_{a*bc}^*(x_0', x_0') = 0$ and $AS_{a*bc}^*(x_0', x_0') = \{ L(\{0\}; 0) \}$. Thus technically, we compute $AS_{a*bc}^*(x_0', Y)$ and $CS_{a*bc}^*(x_0', Y)$, but these are identical to $AS_{a*bc}^*(x_0, Y)$ and $CS_{a*bc}^*(x_0, Y)$.

4.2. **The Period of a Strongly Connected Component**

**Definition 4.1.** The period of a strongly connected component is the gcd (greatest common divisor) of the lengths of all of the cycles in the component. (If a component consists of a single node $v$ with no self-loop $v \rightarrow v$, then assign it a period of 0.)

Note that it does not matter whether we define the period of a strongly connected component (SCC) in terms of all cycles or all simple cycles—the period is the same in either case.

In order to compute the periods of the SCCs we visit the SCCs in $V_a$ in depth-first order, applying the following procedure to each SCC: Suppose we enter an SCC at a node $v$. Give the node $v$ a label $d(v) = 0$, and perform a depth-first traversal of all the edges of the SCC, labeling its nodes and maintaining a count $p$ as follows: When traversing $v \rightarrow x$, if $x$ is not labeled, then give it the label $d(x) = d(v) + w(v \rightarrow x)$, where $w(v \rightarrow x)$ is the weight of edge $v \rightarrow x$. If $x$ is labeled then let $z = d(v) + w(v \rightarrow x) - d(x)$; if $x$ is the first labeled node encountered then set $p = z$, otherwise set $p = \gcd(p, p + z)$. At the end of this traversal, $p$ will be the period of the SCC, as shown by the following lemma.

**Lemma 4.2.** The preceding procedure correctly computes $p$, the period of the strongly connected component on which it is called.

**Proof.** We only reduce $p$ when we have found a cycle of length $z$. Thus, $p$ is the gcd of a number of cycles. These cycles are a subset of the total cycles of the component, and hence $p$ is at least as large as the period. For every edge $v \rightarrow x$, $d(x) \equiv d(v) + w(v \rightarrow x)$ (mod $p$). This implies that every cycle has a length that is a multiple of $p$. Hence, $p$ is larger than the period. Hence, $p$ is equal to the period.

**Definition 4.3.** Let $D^C(x, y) = \{ l : \text{there is a simple path of weight } l \text{ from } x \text{ to } y \text{ in the component } C \}$. Let $S^C(x, y)$ be the simplified distance set from $x$ to $y$ in $C$.

Also, if $p > 0$, the nodes have been partitioned into $p$ equivalence classes, where $u$ and $v$ are in the same equivalence class if and only if $d(u) \equiv d(v)$ (mod $p$). We show that $S^C(x, y) = L(\{d(y) - d(x) \mod p\}; p)$. This will allow us to reduce each component $C_i$ with a nonzero period to a simple representative cycle with $p_i$ nodes, one for each equivalence class.
**Lemma 4.4.** If $p$ is the period of the strongly connected component $C$, and $p \neq 0$, then $(D^C(x, x) \mod p) = \{0\}$.

Proof. Let there be an integer $0 \leq k < p$ such that $k \in (D^C(x, x) \mod p)$. That would mean that there is a cycle of length $ip + k$ for some integer $i \geq 0$. By definition of the period $\gcd(p, ip + k) = p$. This can only happen if $k = 0$. Since $x$ is on some cycle, $D^C(x, x) \neq \emptyset$ and hence $(D^C(x, x) \mod p) = \{0\}$.

**Lemma 4.5.** If $p \neq 0$ is the period of the strongly connected component $C$, then for all $u, v \in C$, if there is a path from $u$ to $v$ of weight $l$, then $(D^C(u, v) \mod p) = \{l \mod p\}$.

Proof. Let there be a path from $x$ to $y$ of length $l$. Hence, the integer $k = l \mod p \in (D^C(x, y) \mod p)$. That is, $l = ip + k$ for some integer $i \geq 0$, and some integer $0 \leq k < p$. Since $C$ is strongly connected there must be a path $P'$ from $y$ to $x$ with length $l'$. From Lemma 4.4, we know that $l' + ip + k \mod p = 0$; hence, $l' \mod p = p - k$. Thus, Lemma 4.4 further implies that all paths from $x$ to $y$ must have lengths $l \equiv k \pmod{p}$. Hence $(D^C(x, y) \mod p) = \{l \mod p\}$.

**Lemma 4.6.** If $p \neq 0$ is the period of the strongly connected component $C$, then for all $x, y \in C$, $(S^C(x, y) \mod p) = (D^C(x, y) \mod p)$.

Proof. Since $D^C(x, y) \subseteq S^C(x, y)$, then $(D^C(x, y) \mod p) \subseteq (S^C(x, y) \mod p)$.

To see that there is no integer $k \in (S^C(x, y) \mod p)$, $k \notin (D^C(x, y) \mod p)$, we assume that there is and will show a contradiction. Corresponding to $k \in (S^C(x, y) \mod p)$, there must be a $k' = k + ip \in S^C(x, y)$. Look at the infinite set of numbers $T = \{k' + jpp' : j \geq 0\}$. Clearly $T \subseteq S^C(x, y)$. Also, for all $t \in T$, we can see that $t \notin D^C(x, y)$, because otherwise we would have $k \in (D^C(x, y) \mod p)$. Thus, the infinite set $T \subseteq S^C(x, y) \setminus D^C(x, y)$. This is impossible by the properties of the periodic set $S^C(x, y)$.

**Lemma 4.7.** If $p \neq 0$ is the period of the strongly connected component $C$, then for all $x, y \in C$, $S^C(x, y) = L(D^C(x, y) \mod p; p)$.

Proof. So we have shown that $(S^C(x, y) \mod p) = (D^C(x, y) \mod p)$. To see that $S^C(x, y) = L(D^C(x, y) \mod p; p)$, we just need to look at Lemma 3.7.

**Lemma 4.8.** If $p = 0$, then if $x$ is the node in the strong component, $S^C(x, x) = \{0\} = L(\{0\}; \emptyset) = L(\{0\}; 0)$.

Proof. Immediate from the definition of the period of a strong component and the fact that $p = 0$.

All this allows us to replace each SCC with a representative cycle without losing any necessary information. Figure 4 shows the result of this transformation on an isolated strongly connected component. We would like to further reduce each representative cycle to a single representative node without losing any necessary
information. The difficulty results from edges connecting nodes of different strongly connected components.

4.3. Simplifying Intercomponent Edges

To reduce each component to a single node, we add weighted edges to the graph, where the weight represents the "length" of the edge. If there is an $a$-edge from some node $v$ of class $c_1$ in a component $C$ with period $p_1 > 0$ to some node $x$ of class $c_2$ in a component $C'$ with period $p_2 > 0$, replace it by an auxiliary edge of weight $c_1 + w(v \rightarrow x) - c_2 \mod p_1$ from the class 0 representative node of $C$ to the class 0 representative node of $C'$. (If the edge is a $b$-edge, replace it with an edge of weight $c_1 + w(v \rightarrow x)$ that still goes to $x$.) If performing these replacements causes duplicate edges, collapse them together, giving the resulting edge a label that is the set of the previous labels.

If $C$ has period $p_1 = 0$, then it must consist of a single node, say $v$. If there is an edge from $v$ to a node $x$ in a class $c_2$ of a component $C'$ with period $p_2 > 0$, replace it by an edge of weight $w(v \rightarrow x) + c_2$ from $v$ to the class 0 representative of $C'$.

If $C$ has period $p_1 > 0$, but $C'$ has period $p_2 = 0$, then $C'$ must consist of a single node, say $x$. If there is an edge from $v$ in a class $c_1$ of $C$ to $x$, replace it by an edge of weight $c_1 + w(v \rightarrow x)$ from the class 0 representative of $C$ to $x$.

This leaves a DAG whose nodes are either cycle representatives, or nodes with no self-loops. Associated with each cycle representative $v$ is $p(v)$, the period of the SCC that $v$ represents. Associated with each edge $e = v \rightarrow w$, where $v$ is a cycle representative, in a set $W(e)$ of weights in the range $0 \cdots p(v) - 1$. Figure 5 gives an example of this transformation on two strongly connected components.

**Lemma 4.9.** Let $CD(x, y)$ be the cyclic distance set for $(x, y)$ in some graph $G$ where we have collapsed SCCs into simple cycles, and let $G'$ be the result of collapsing the simple cycles into of $G$ into representative nodes, and let $CD'(x, y)$ be the corresponding distance set in $G'$. Then if $x$ and $y$ are designated class 0 representatives for their respective SCCs, $R(CD(x, y)) \subseteq R(CD'(x, y))$ in $G$, and $R(CD'(x, y)) \setminus R(CD(x, y))$ is a finite set.

**Proof.** If $p(x) = 0$ or $p(y) = 0$, the lemma is obvious.

Assume that we replaced an edge from a node $v$ of class $c_1$ in $x$'s cycle to a node
becomes

\[ L = c_1 + i \cdot p(x) + w(v \rightarrow w) + (p(y) - c_2) + j \cdot p(y) \]

for \( i, j \geq 0 \). We replace the edge with one of length \( c_1 + w(v \rightarrow w) - c_2 \mod p_1 \) from \( x \) to \( y \), so the paths from \( x \) to \( y \) over the replacement edge are of lengths

\[ L' = i \cdot p(x) + (c_1 + w(v \rightarrow w) - c_2 \mod p(x)) + j \cdot p(y) \]

for \( i, j \geq 0 \). Since, \((c_1 + w(v \rightarrow w) - c_2 \mod p(x))\) is no larger than \((c_1 + w(v \rightarrow w) + (p(y) - c_2))\) and

\[
(c_1 + w(v \rightarrow w) - c_2 \mod p(x)) = (c_1 + w(v \rightarrow w) + (p(y) - c_2)) \mod \gcd(p(x), p(y)),
\]

then \( L \leq L' \), and they differ by only a finite number of elements.  

### 4.4. Computing Simplified Distance Sets

Now we scan the nodes of the DAG in topological order, computing the acyclic distance sets and the simplified byclic distance sets for each node as we go. To do this, we use the relationship

\[
D(x_0, w) = \bigcup_{v \in \text{Pred}(w)} D(x_0, v) + W(v \rightarrow w) + D(w, w).
\]

The actual computation is complicated by two issues.
1. The node $w$ can either be a cycle representative or a single node with $p(w) = 0$.

2. We must maintain both the cyclic sets $\text{CS}(x_0, w)$ and the acyclic sets $\text{AS}(x_0, w)$.

First, we outline the straightforward way to do the computation. Next, we give a more efficient approach. In this section, we consider the computation in step 2 of Fig. 2; in subsection 4.5 we consider the modifications necessary for step 6.

**Straightforward Approach**

Suppose that at some point during the algorithm we wish to compute the simplified distance set for a cycle representative node $w$. Let $v_1, ..., v_d$ be the predecessors of $w$. At this point the acyclic distance sets $\text{AS}(x_0, v_i)$ and the simplified cyclic distance sets $\text{CS}(x_0, v_i)$, where $1 \leq i \leq d$, have already been computed. There are two cases to consider, depending on whether the node $w$ is in a component with cycle $p(w) = 0$ or $p(w) \neq 0$.

**Case 1.** The node $w$ has period $p(w) = 0$. Here, $D(w, w) = L(\{0\}; 0) = \{0\}$, so we would just need to compute the nonsimplified sets as

$$
\text{AD}(x_0, w) = \bigcup_{1 \leq i \leq d} \text{AD}(x_0, v_i) + W(v_i \rightarrow w)
$$

$$
\text{CD}(x_0, w) = \bigcup_{1 \leq i \leq d} \text{CD}(x_0, v_i) \cup W(v_i \rightarrow w),
$$

where, again, the $v_i$ are the predecessors of $w$. As $p(w) = 0$, passing through $w$ will not introduce cycles, so we may consider the sets $\text{CS}(x_0, w)$ and $\text{AS}(x_0, w)$ separately. By definition of $R(\cdot)$ and Lemmas 3.9 and 3.10:

$$
\text{AS}(x_0, w) = \bigcup_{1 \leq i \leq d} \text{AS}(x_0, v_i) + W(v_i \rightarrow w)
$$

$$
\text{CS}(x_0, w) = R(\text{CD}(x_0, w))
$$

$$
\begin{align*}
&= R \left( \bigcup_{1 \leq i \leq d} \text{CD}(x_0, v_i) + W(v_i \rightarrow w) \right) \\
&= R \left( \bigcup_{1 \leq i \leq d} R(\text{CD}(x_0, v_i)) + W(v_i \rightarrow w) \right) \\
&= R \left( \bigcup_{1 \leq i \leq d} \text{CS}(x_0, v_i) + W(v_i \rightarrow w) \right).
\end{align*}
$$

**Case 2.** The node $w$ has period $p(w) = p \neq 0$. This means that the set $\text{CD}(w, w)$ will be $L(\{0\}, p)$. 

Since $p \neq 0$, we know that every path through $w$ will contain a cycle, so by definition of $\text{AD}$ and $\text{AS}$, both will be empty.

\begin{align*}
\text{AD}(x_0, w) &= \emptyset \\
\text{AS}(x_0, w) &= \emptyset.
\end{align*}

(3)

Since the paths that had not previously hit a cycle now have hit one, we have that

\begin{equation*}
\text{CD}(x_0, w) = \bigcup_{1 \leq i \leq d} (\text{CD}(x_0, v_i) \cup \text{AD}(x_0, v_i)) + W(v_i \rightarrow w) + \text{CD}(w, w)
\end{equation*}

By definition of $R(\ )$ and Lemmas 3.9 and 3.10:

\begin{align*}
\text{CS}(x_0, w) &= R(\text{CD}(x_0, w)) \\
&= R \left( \bigcup_{1 \leq i \leq d} (\text{CD}(x_0, v_i) \cup \text{AD}(x_0, v_i)) + W(v_i \rightarrow w) + \text{CD}(w, w) \right) \\
&= R \left( \bigcup_{1 \leq i \leq d} (R(\text{CD}(x_0, v_i)) \cup \text{AS}(x_0, v_i)) + W(v_i \rightarrow w) + R(\text{CS}(w, w)) \right) \\
&= R \left( \bigcup_{1 \leq i \leq d} (\text{CS}(x_0, v_i) \cup \text{AS}(x_0; v_i)) + W(v_i \rightarrow w) + \text{CS}(w, w) \right) \\
&= R \left( \bigcup_{1 \leq i \leq d} \text{CS}(x_0, v_i) + W(v_i \rightarrow w) + \text{CS}(w, w) \right).
\end{align*}

(4)

**Efficient Approach For Step 2**

As the operator $R(\ )$ commutes with set addition and union, there is a choice about when it should be applied during the topological traversal of the strongly connected components. One extreme, outlined above, is to apply $R(\ )$ as soon as possible, that is, whenever one passes through a strongly connected component with a nonzero period. Another extreme would be to defer the application of $R(\ )$ until the end of the traversal. That approach actually computes the distance sets rather than the simplified distance sets, and simplifies as the last step. The most efficient approach lies between the two extremes, and will be explained in detail below. This approach regulates applications of $R(\ )$ in such a way that at most $O(|V|)$ storage is required per AS or CS set.

Again, suppose that at the current point of the algorithm we wish to compute the simplified distance set for a node $w$. Let $v_1, \ldots, v_d$ be the predecessors of $w$. At this point the simplified distance sets for $v_i$, where $1 \leq i \leq d$, have already been computed. There are two cases to consider.

**Case 1.** The node $w$ has period $p(w) = 0$. We use Eq. (1) for AS, but instead of Eq. (2) we use

\begin{equation*}
\text{CS}(x_0, w) = \bigcup_{1 \leq i \leq d} \text{CS}(x_0, v_i) + W(v_i \rightarrow w).
\end{equation*}

(5)
The set $\text{CS}(x_0, w)$ is the same either way, but it is represented differently. Using Eq. (5), the set is represented as a union of periodic sets rather than a single periodic set. We choose to do this because applying $R(\ )$ now may cause the amount of storage required to store CS to blow up too large.

Unfortunately, deferring the application of $R(\ )$ means that the sets $\text{CS}(x_0, v_i)$ may also be represented as unions of periodic sets instead of simply periodic sets. This means that the following explanation is doomed to being convoluted.

Let $\text{CS}(x_0, v_i) = \bigcup_{1 \leq j \leq s_i} L(C_{i,j} ; p_{i,j})$. First compute

$$\text{CS}_{i} = \bigcup_{1 \leq j \leq s_i} L(C_{i,j} + W(v_i \rightarrow w) \mod p_{i,j} ; p_{i,j})$$

(6)

for each $i$. The sets $\text{CS}_{i}$ are still unions of periodic sets. We know that $\text{CS}(x_0, w) = \bigcup_{i} \text{CS}_{i}$. The set $\text{CS}(x_0, w)$ will be a union of periodic sets. There is however one simplification that we do make at this point: If there are periodic sets $L(C_{1}; p)$ and $L(C_{2}; p)$ with the same period, we replace them with a single periodic set $L(C_{1} \cup C_{2}; p)$.

Figure 6 shows an example of this computation for $d = 2$.

In step 2, the AS sets are aperiodic sets and we just represent them as one single aperiodic set $L(C_{i}; 0)$. In that case, for each $v_i$ the set $\text{AS}(x_0, v_i)$ will be an aperiodic set $L(C_{i}; 0)$. The set $\text{AS}(x_0, w)$ can be computed by computing the set sums $C_{i} + W(v_i \rightarrow w)$, for $1 \leq i \leq d$, and taking the union of the results. In symbols, we have

$$\text{AS}(x_0, w) = L \left( \bigcup_{v_i} (C_{i} + W(v_i \rightarrow w)); \emptyset \right).$$

(7)

Case 2. The node $w$ has period $p(w) \neq 0$. We already know from Eq. (3) that $\text{AS}(x_0, w) = \emptyset$. We just need to show how to efficiently compute $\text{CS}(x_0, w)$. From Eq. (4) we can see that, for each $i$, the contribution to $\text{CS}(x_0, w)$ from node $v_i$ will consist of two parts: the contribution from $\text{AS}(x_0, v_i)$ and the contribution from $\text{CS}(x_0, v_i)$.

$$\text{CS}(x_0, v_1) = L\{(1, 3), 5\}$$

$\text{CS}(x_0, v_2) = L\{(2, 3), 5\} \cup L\{(1), 2\}$

$$\text{CS}(x_0, w) = L\{(0, 2, 3, 4), 5\} \cup L\{(0), 2\}$$

Fig. 6. Combining sets when $p(w) = 0$. 
We begin with the contribution from CS(x0, vi), using the sets CS′ defined in Eq. (6). Now we need to add to each of these the periodic set CS(w, w). In the following we use the shorthand \( \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} \text{gcd}(p_{i,j}, p(w)) = g_{i,j,w} \).

The sum of two periodic sets \( L(C_i; p_1) \) and \( L(C_j; p_2) \) is the set \( L(C_i + C_j; \{p_1, p_2\}) \). Note that if \( p_1 \neq p_2 \), the resulting set is not periodic. Recalling that \( CS(w, w) = JW_0; \text{gcd}(p(w)) \)

we may define the set \( CS'' = \bigcup_{1 \leq i \leq d} \text{CS}' + CS(w, w) \)

\[ = \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} L(C_i; p_{i,j}) + L(\emptyset; p(w)) \]

\[ = \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} L(C_{i,j}; \{p_{i,j}, p(w)\}) \]

The contribution to CS(x0, w) due to the cyclic distance sets of the predecessors of w is just

\[ R(CS'') = M \left( \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} B(L(C_{i,j}; \{p_{i,j}, p(w)\})) \right) \]

\[ = M \left( \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} L(C_{i,j} \mod g_{i,j,w}; g_{i,j,w}) \right) \]

\[ = L \left( \bigcup_{1 \leq i \leq d} \bigcup_{1 \leq j \leq s_i} L(C_{i,j} \mod g_{i,j,w}; g_{i,j,w}) \mod g; g \right), \quad (8) \]

where \( g = \text{lcm}(\text{gcd}(p_{1,1}, p(w)), \ldots, \text{gcd}(p_{d,s_d}, p(w))) \). We compute g as follows.

\( g = 1 \)

\( \text{for } i = 1 \text{ to } d \) do

\( \text{for } j = 1 \text{ to } s_i \) do

begin

\( g' = g_{i,j,w} / \text{gcd}(p_{i,j}, g) \);

\( g = g \cdot g' \)

end.

In step 2 the sets AS(x0, vi) are each just finite sets of constants. Let AS' be the union of adding the set AS(x0, vi) to the set \( W(v_i \rightarrow w) \) for each vi. That is, \( AS' = \bigcup_{1 \leq i \leq d} AS(x_0, v_i) + W(v_i \rightarrow w) \). Adding AS' to CS(w, w) will be the contribution to CS(x0, w) due to the acyclic distance sets of the predecessors of w. In symbols, this is just the set \( AS'' = AS' + L(\emptyset; p) = L(AS'; p) \) so

\[ R(AS'') = L(AS' \mod p; p). \quad (9) \]

That is, in this case, if \( AS' \neq \emptyset \) then \( g = p \).
In total, the set $CS(x_0, w)$ is the union of the contribution due to the cyclic distance sets and the contribution due to the acyclic distance sets of the predecessors of $w$. We have

$$CS(x_0, w) = R(R(AS'') \cup R(CS'')) = L((R(AS'') \cup R(CS'')) \mod g'; g'),$$

where $g' = g$ if $AS' = \emptyset$ and $g' = p$ otherwise, and $R(AS'')$ is as defined in Eq. (9) and $R(CS'')$ is as defined in Eq. (8). Figure 7 shows an example of this computation for $d = 2$.

4.5. Modifications for Step 6

When we compute $AS_{a,b,c}^*(x_0, y)$ and $CS_{a,b,c}^*(x_0, y)$ in step 6 of Fig. 2, a few simple changes to the above method for step 2 are required. First, all references to paths of $a$-edges should now refer to paths of $c$-edges. Second, instead of a single fake node, we use the nodes in $V'$. Each node $x \in V'$ is initialized so that $CS_{a,b,c}^*(x_0, x) = \emptyset$ and $AS_{a,b,c}^*(x_0, x) = CS_{a,c}^*(x_0, x)$.

More interestingly, since the $CS$ sets are, in effect, the differences of semilinear sets, they may contain negative numbers. That is, the $L(C; p)$ elements of $CS_{a,b,c}^*(x_0, y)$ now represent the sets generated by also allowing negative multiples of $p$—the $\lambda$'s are allowed to range over the negative integers also. This does not actually affect the implementation of step 6 in any way.

A more major change is that in step 6; the $AS$ sets are now periodic. Fortunately, this saves us from having to allow negative numbers in any $C$. It does not, however, save us from a few implementation changes. When computing $AS$ in Case 1, we actually must use the same method described for $CS$ in Eq. (5) and (6), with $CS$ replaced by $AS$ in those equations. For Case 2, we need to compute $R(AS'')$ in the same manner as $R(CS'')$ and then to get $CS(x_0, w)$ we just compute $R( )$ applied to the union of the two results; there is a slightly more efficient but harder to describe method of combining these two computations into one.

![Figure 7](image-url)
4.6. An Example

We consider the query \( t(x, Y) \) on the example graph of Fig. 8. In that graph, the part labeled by \( A \) (everything to the left of the edge from \( p \) to \( q \)) corresponds to \( a \)-edges, the part labeled by \( B \) (the edge from \( p \) to \( q \)) corresponds to \( b \)-edges, and the part labeled by \( C \) (everything to the right of the edge from \( p \) to \( q \)) corresponds to \( c \)-edges.

The first step is to preprocess the graph, collapsing the strongly connected components into single nodes. The result of performing this reduction appears in Fig. 9. In that graph, nodes labeled by capital letters correspond to collapsed strongly connected components, while nodes corresponding to original nodes of the graph are labeled by lowercase letters. If an edge has no label, it is of weight 1; if a node has no period label, it has a period of 0 (no cycle.)

Next, run the algorithm of Fig. 2. Step 1 identifies all nodes of the reduced graph, since they are all reachable from \( x \). Step 2 computes the simplified distance sets as given in Table 1.

Steps 3, 4, and 5 in this case identify all nodes in the "C" portion of the graph. Step 6 computes the simplified distance sets as given in Table 2.
TABLE I
Distance Sets Computed in Step 2

<table>
<thead>
<tr>
<th>Distance Set</th>
<th>Computed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{AS}_*(x, x) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, x) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, A) )</td>
<td>( L({1}, 2) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, A) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, B) )</td>
<td>( L({1}, 2) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, B) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, g) )</td>
<td>( L({0}, 2) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, g) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, C) )</td>
<td>( L({1}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, C) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, II) )</td>
<td>( L({1}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, II) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, n) )</td>
<td>( L({2}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, n) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, m) )</td>
<td>( L({1, 3}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, m) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, o) )</td>
<td>( L({0, 2}, 4) \cup L({2}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, o) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, p) )</td>
<td>( L({1}, 2) \cup L({1, 3}, 4) )</td>
</tr>
</tbody>
</table>

Step 7 of the algorithm in Fig. 2 determines the answer nodes. Immediately we see that \( v \) is an answer, since zero appears in its simplified cyclic distance set. For nodes \( D \) and \( E \), one must recall that these nodes actually represent strongly connected components of the original graph. Since the simplified cyclic distance set for \( D \) is \( L(\{1\}, 2) \), zero will be in the reachability set for nodes of \( D \) corresponding to levels 1 and 3, which is just the nodes \( s \) and \( t \). Since the simplified cyclic distance set for \( E \) has period 1, all nodes of \( E \) (\( w, y, \) and \( z \)) are answers. To summarize, the set of answers to \( t(x, Y) \) is \( \{ s, t, v, w, y, z \} \).

TABLE II
Distance Sets Computed in Step 6

<table>
<thead>
<tr>
<th>Distance Set</th>
<th>Computed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{AS}_*(x, p) )</td>
<td>( L({1}, 2) \cup L({1, 3}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, p) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, q) )</td>
<td>( L({0}, 2) \cup L({0, 2}, 4) )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, q) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, D) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, D) )</td>
<td>( L({1}, 2) )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, v) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, v) )</td>
<td>( L({0}, 2) )</td>
</tr>
<tr>
<td>( \text{AS}_*(x, E) )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \text{CS}_*(x, E) )</td>
<td>( L({0}, 1) )</td>
</tr>
</tbody>
</table>
5. Timing Analysis

The following refers to the steps in the algorithm in Fig. 2.
Step 1 of the algorithm in Fig. 2 takes \( O(|E_a|) \) time. Step 3 takes \( O(|V_a|) \) time. Step 4 takes \( O(|V_a| + |E_a|) \) time. Step 5 takes \( O(|V_b| + |E_c|) \) time.

The following analysis holds for both steps 2 and 6. For step 2, substitute \( |V| = |V_a| \) and \( |E| = |E_a| + |E_b| \). For step 6, substitute \( |V| = |V_a| + |V_c| \) and \( |E| = |E_b| + |E_c| \).

In step 2b we simply use the standard algorithm [Tar72], which finds the strongly connected components in \( O(|E|) \) time.

The method in Subsection 4.2 computes the period of a component in time proportional to doing a depth-first search of the component. Hence step 2c can be done in \( O(|E|) \).

If we follow the method in Subsection 4.3 to perform step 2d, we never need to look at any edge twice. There is only a constant amount of work per edge, so overall this takes \( O(|E|) \) time.

5.1. Timing Analysis of Step 2e

To analyze the time for computing the simplified distance sets, we must consider data structures. We represent a \( W(e) \) as a linked list of increasing integers. A simplified distance set (a CS or AS) is represented as a linked list of \( L(C; p) \) elements where the elements are in increasing order according to \( p \). A \( L(C; p) \) is represented as a record with a field for the number \( p \) and a field with a linked list of increasing integers for \( C \).

For any cycle representative \( w \) let \( T(w) \) be the sum of \( \max(p(v), 1) \) over all cycle representatives \( v \) that are topological predecessors of \( w \) (\( w \) is a predecessor of itself). We show that the sets \( CS(x_a, w) \) and \( AS(x_a, w) \), which are made up of \( k \) different \( L(C; p) \) terms, are such that \( \sum_{i=1}^{k} |C_i| + \sum_{i=1}^{k} |C_i| \leq T(w) \leq |V| \). We can see this by looking at the two cases.

1. If \( p(w) = 0 \), note that in performing the unions to compute \( CS(x_0, w) \) and \( AS(x_0, w) \), we only combine two \( L(C; p) \) elements if they have equal \( p \)-values. Thus each \( p \)-value such that \( p > 0 \) of \( CS(x_0, w) \) and \( AS(x_0, w) \) can be "charged" to different predecessor node \( v \) with \( p \leq p(v) \). Furthermore, the cost of a term \( L(C; 0) \) can be charged to \( |C| \) predecessors nodes \( v \) with \( p(v) = 0 \). Hence, the sum of the \( p \)-values for any pair of sets \( CS(x_0, w) \) and \( AS(x_0, w) \) with \( p(w) = 0 \) is bounded by \( T(w) \).

2. If \( p(w) \neq 0 \), the set \( AS(x_0, w) = \emptyset \) and the set \( CS(x_0, w) \) is a single \( L(C; p) \) with \( p \leq p(w) \). Hence, \( |C| \leq p(w) \leq T(w) \).

This bound on the size of the \( L(C; p) \) sets that can arise in the course of the algorithm provides the following bounds on the timings of operations on those sets.

Computing a single sum \( L(C; p) + W(e) \) can be done in time \( O(|C| \cdot |W(e)| + p) \), since the result will be of the form \( L(C'; p) \). Thus, the total cost of doing these set
sums for a single edge $e$ is bounded by $O(|V| \cdot |W(e)|)$. Hence the total cost of performing the set sums over all the edges is $O(|V| \cdot |E|)$.

The cost of taking the union of $k$ different $L(C; p)$ elements with the same $p$ is $|C_1| + |C_2| + \cdots + |C_k| + p \leq (k + 1) \cdot p$. The sum of these for a single cycle representative $v$ is $O(|V| \cdot d_{in}(v))$. Hence the total cost the unions for the whole DAG is $O(|V| \cdot |E|)$.

The time to perform an $S \bmod g$ operation is bounded by $g$ times the sum of the $|C|$ elements, which in turn is bounded by $|V| \cdot g$. So these operations contribute at most $O(|V|^2)$ to the total.

5.2. Total Time

Thus, steps 2 and 6 can be done in $O(|V| \cdot |E|)$ time.

By step 7, we have computed the sets $CS_{a^*bc^*}(x_0, y)$ for all $y$ that are cycle representatives. For a nonrepresentative node $v$ in equivalence class $c$ of the component with $y$ as its representative and with period $p(y), 0 \in CS_{a^*bc^*}(x_0, v) \iff p(y) - c \in CS_{a^*bc^*}(x_0, y)$. This step can be done in $O(|V_c| \cdot (|V_c| + |V_a|))$ time.

Adding it all up, the whole algorithm requires $O(ne)$ time.

6. Conclusion

By generalizing the notion of edges in graphs to hyper-edges in hyper-graphs our method can be extended to handle more general linear recursions. Note, however, that any extension of counting methods to multiple recursive rules will be extremely inefficient on certain databases. This is because counting methods record (tuple, derivation) pairs, and in any multiple recursive rule definition there can be an exponential number of derivations for some tuples.

References


[GSSS87] G. Grahne, S. Sippu, and E. Soisalon-Soininen, Efficient evaluation for a subset of recur-
A CYCLIC BINARY QUERY


