Generating Elementary Cycles in Maximal Reducible Flowgraphs[†]

Kuo-Hua Kao[1] Jou-Ming Chang[1,2,‡] Yuwen Cheng[3]

[1] Graduate Institute of Business Administration, National Taipei College of Business, Taipei, Taiwan, ROC

[2] Department of Information Management, National Taipei College of Business, Taipei, Taiwan, ROC

[3] Department of Mathematics, National Taitung University, Taitung, Taiwan, ROC

Abstract

Reducible flowgraphs serve as useful vehicle in data flow analysis of computer programs. A reducible flowgraph is maximal (MRF for short) if no more edge can be added without violating reducibility. Based on a decomposition theorem of MRFs proposed in [Congr. Numer. 139 (1999) 9–20], a formula for counting the number of elementary cycles in MRFs is derived. Moreover, we propose an algorithm for generating all elementary cycles of an MRF when its corresponding decomposition tree is given.

Keywords: elementary cycles; reducible flowgraphs; graph decomposition; enumeration algorithms;

1. Introduction

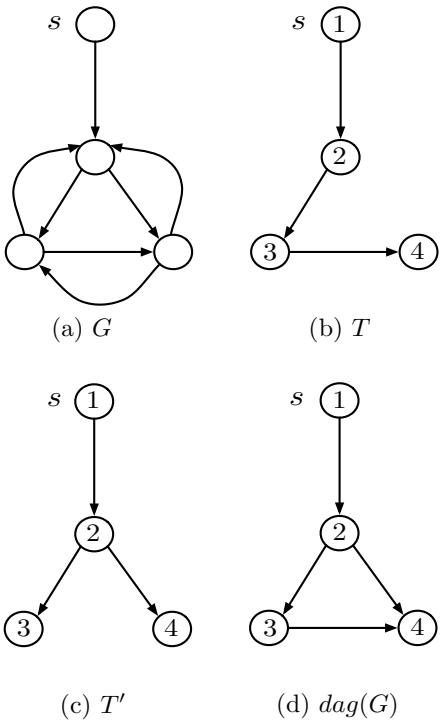
Flowgraphs occur naturally in connection with control flow and code optimization of computer programs [1, 2]. A *flowgraph* $G = (V, E, s)$ is a digraph (V, E) with a distinguished vertex s, called its *source*, such that every vertex of G is reachable from s. In particular, G is called a *directed rooted tree* with s as its root if $|E| = |V| - 1$. Also, $(\{s\}, \emptyset, s)$ is called a *trivial* flowgraph. For a given flowgraph $G = (V, E, s)$, if a spanning subgraph $T = (V, E', s)$ of G is a directed rooted tree with s as its root, we simply call T a *directed rooted spanning tree* (DRST). Further, T is a *depth-first search tree* (DFST) if T is a DRST and there is a preorder numbering on V such that for every (directed) edge $(u, v) \in E \setminus E'$, either v is a descendant of u in T, or the preorder number of v is less than the preorder number of u. Hereafter, an edge $e = (u, v)$ in G is called a *back edge* if $e \in E \setminus E'$ and there is a directed path from v to u in T, and e is called a *forward edge* otherwise. Note that the classification of edges depends on the spanning tree T which can be obtained from a depth-first search (DFS) on G starting at s. Let $G = (V, E, s)$ be a flowgraph and B the back edge set of G with respect to a DFST T. Then the spanning subgraph $dag_T(G) = (V, E - B, s)$ is called the *directed acyclic graph* of G with respect to T. We omit the subscript T in the notation $dag_T(G)$ when no ambiguity arises. Obviously, $dag(G)$ is acyclic and adding any edge of B into $dag(G)$ will create a cycle.

For example, Figure 1(a) shows a flowgraph G. Two DRSTs of G are shown in Figure 1(b) and Figure 1(c), respectively. In T and T', vertices are labeled according to a preorder numbering. It is easy to check that T is a DFST, and thus with respect to T, G has back edge set $\{(3, 2), (4, 2), (4, 3)\}$ and forward edge set $\{(1, 2), (2, 3), (2, 4), (3, 4)\}$. However, T' is not a DFST since there is an edge $(3, 4)$ in G such that node 4 is not a descendant of node 3 in T'. Also, the directed acyclic graph of G with respect to T is shown in Figure 1(d).

For a flowgraph $G = (V, E, s)$ and $u, v \in V$, we say that u *dominates* v (or v is dominated by u) if every path from s to v passes through u. If u dominates v and $u \neq v$ then u is a *dominator* of v. In particular, u is the *immediate dominator* of v, denoted by $u = idom(v)$, if u is a dominator of v and every other dominator of v dominates u. The

[†]This work was supported by the National Science Council of Taiwan under the contract NSC94-2115-M-141-001.

[‡]All correspondence should be addressed to Professor Jou-Ming Chang, Department of Information Management, National Taipei College of Business, No. 321, Section 1, Tsi–Nan Road, Taipei, Taiwan. (Email: spade@mail.ntcb.edu.tw).



(a) G (b) T

(c) T' (d) $dag(G)$

Figure 1: (a) A flowgraph G; (b) A DFST T of G; (c) A DRST T' of G that it is not a DFST; (d) The $dag(G)$ with respect to T.

dominance relation in a flowgraph $G = (V, E, s)$ is a partial order on V, and its Hasse Diagram is a rooted tree, called the *dominator tree* of G. For example, consider the flowgraph G shown in Figure 1(a) again. Then the dominator tree of G is isomorphic to T' shown in Figure 1(c).

In most of characterizations of flowgraphs, a property called *reducibility* due to [4, 5] is the most important. In the flowgraph representations of structured programs, there are no jumps into the middle of the loops from outside. This suggests that all the loops are single-entry and such a group of nodes in a loop can be considered as a single node during the data flow analysis. Therefore, we can "reduce" a group of nodes into a single node using successive applications of some well defined transformations [5]. The flowgraphs which can be reduced so to a single node are called *reducible flowgraphs*. Much of the researches in data flow analysis are centered around this class of graphs. The following are equivalent definitions of reducible flowgraphs [4, 5]:

(1) $G = (V, E, s)$ is a reducible flowgraph.
(2) Every DFS on G starting at s determines the same back edge set B.
(3) The $dag(G) = (V, E - B, s)$ is unique.
(4) For every back edge $(u, v) \in B$, v dominates u.
(5) G does not contain the subflowgraph $SP(s, x, y, z)$ depicted as Figure 2.
(6) Every cycle of G has a vertex which dominates the other vertices of the cycle.

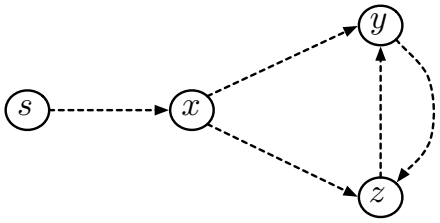


Figure 2: The forbidden subflowgraph $SP(s, x, y, z)$, where a dashed line from u to v represents that there is a path from u to v in G.

For a digraph $G = (V, E)$, a *path* of length k is a set of edges of the form $\{(v_i, v_{i+1}) \in E \mid i = 0, 1, \ldots, k-1\}$, a *cycle* is a path with $v_0 = v_k$. A path is *elementary* if it encounters no vertex twice. An *elementary cycle* is similarly defined. Researches have been done on algorithms to list up all elementary cycles in directed as well as undirected graphs [3, 7, 9, 10, 13, 14]. In this paper, we study the cycle structures on an interesting subclass of reducible flowgraphs, called the maximal reducible flowgraphs (referred as MRFs for short). Based on a decomposition theorem, we deduce a formula for counting the number of elementary cycles in MRFs. Moreover, we also propose an algorithm for enumerating all elementary cycles of an MRF when its corresponding decomposition tree is available.

2. Maximal reducible flowgraphs and their decompositions

In this section, we review the relevant properties of MRFs. A reducible flowgraph is *maximal* if it becomes irreducible by the addition of an extra edge. Formally, $G = (V, E, s)$ is an MRF if G is reducible and $G + (x, y)$ is not reducible for every pair of vertices $x, y \in V$ such that $x \neq y$ and $(x, y) \notin E$. A trivial flowgraph is also considered as an MRF. The significance of MRFs lies in the fact that they can be used to derive or prove upper (or lower) bounds on the properties of reducible flowgraphs.

In [11], the following binary operation is defined on flowgraphs for characterizing MRFs. Let $G_1 = (V_1, E_1, s_1)$ and $G_2 = (V_2, E_2, s_2)$ be two flowgraphs such that $V_1 \cap V_2 = \emptyset$. The *product* $\otimes$ of G_1 and G_2

is the flowgraph defined as

$$G_1 \otimes G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup (V_1 \times \{s_2\}) \cup (V_2 \times \{s_1\}), s_1),$$

where all vertices and edges of G_1 and G_2 are preserved under $\otimes$, and the new edges are added from every vertex of G_1 to s_2 (i.e., the forward edges) and from every vertex of G_2 to s_1 (i.e., the back edges). Using the definition, Vernet and Markenzon [11] characterize MRF as follows:

Theorem 1. (Decomposition Theorem) *A non-trivial flowgraph G is maximal reducible if and only if there exist unique maximal reducible flowgraphs G_1 and G_2 such that $G = G_1 \otimes G_2$.*

The above characterization suggests that there is a tree structure associated with the decomposition process of an MRF. Let $G = (V, E, s)$ be an MRF. The *decomposition tree* of G, denoted by $DT(G)$, is an extended binary tree and it can be recursively constructed as follows:

```
If G is a trivial flowgraph then
    DT(G) := G
else
    assume G = G1 ⊗ G2
    create a root for DT(G)
    set DT(G1) as the left subtree of the root
    set DT(G2) as the right subtree of the root
endif
```

Figure 3(a) shows an MRF and its decomposition process, and Figure 3(b) shows the corresponding decomposition tree. Conversely, given an extended binary tree T, it is possible to construct a unique MRF G by traversing T in postorder such that $DT(G) = T$. Based on the decomposition theorem, several graph problems in MRFs are efficiently solved [12], including hamiltonian paths and cycles, testing isomorphism, and minimum cardinality feedback edge sets.

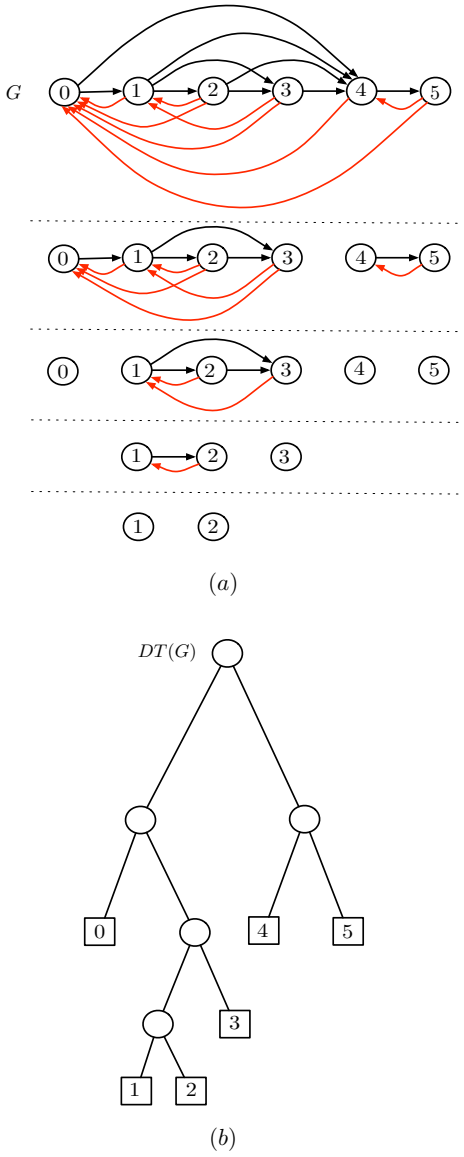


Figure 3: An MRF, its decomposition and the associated decomposition tree.

3. Counting the number of elementary cycles in MRFs

Throughout the rest we assume that G is an MRF. From the structure of decomposition process, we can see that every internal node (respectively, leaf) of $DT(G)$ corresponds to a nontrivial (respectively, trivial) subgraph of G. Thus, for each node $x \in DT(G)$, we write $G(x)$ as its corresponding subgraph of G. Also, for each internal node x, we use x_L and x_R to denote the left child and the right child of x, respectively. By decomposition theorem, $G(x) = G(x_L) \otimes G(x_R)$. Let s_x be the source of $G(x)$, and we denote $p(G(x), v)$ the number of distinct elementary paths from s_x to a vertex v in $G(x)$. For convenience, we treat a single vertex as a path with length 0. Thus, $p(G(x), s_x) = 1$. We now define $p(x) = \sum_{v \in G(x)} p(G(x), v)$ as the total number of elementary paths starting from s_x in $G(x)$.

Lemma 1. *For each node $x \in DT(G)$, $p(x)$ can be computed recursively as follows:*

$$p(x) = \begin{cases} 1, & \text{if } x \text{ is a leaf in } DT(G) \\ p(x_L) \cdot \left[1 + p(x_R)\right], & \text{otherwise.} \end{cases}$$

Proof. We may assume that x is an internal node. Observe that (1) if $v \in G(x_L)$, then every elementary path from s_x to v cannot pass through $G(x_R)$;

(2) if $v \in G(x_R)$, then every elementary path from s_x to v must pass through the source of $G(x_R)$ (i.e., the source of $G(x_R)$ dominates v in $G(x)$). Moreover, every vertex of $G(x_L)$ is connected by an edge to the source of $G(x_R)$. Thus,

$$p(G(x),v)=\begin{cases} p\left(G(x_L),v\right), & \text{if } v \in G(x_L) \\ \left[\sum_{u\in G(x_L)} p\left(G(x_L),u\right)\right]\cdot p\left(G(x_R),v\right), & \text{if } v \in G(x_R). \end{cases}$$

Consequently,

$$\begin{aligned} p(x) &= \sum_{v\in G(x_L)} p\left(G(x_L),v\right) + \\ &\quad \sum_{v\in G(x_R)} \left[\sum_{u\in G(x_L)} p\left(G(x_L),u\right)\right]\cdot p\left(G(x_R),v\right) \\ &= \sum_{v\in G(x_L)} p\left(G(x_L),v\right) + \\ &\quad \left[\sum_{u\in G(x_L)} p\left(G(x_L),u\right)\right]\cdot\left[\sum_{v\in G(x_R)} p\left(G(x_R),v\right)\right] \\ &= \sum_{v\in G(x_L)} p\left(G(x_L),v\right)\cdot\left[1+\sum_{v\in G(x_R)} p\left(G(x_R),v\right)\right] \\ &= p(x_L)\cdot[1+p(x_R)] \end{aligned}$$

□

For each node $x \in DT(G)$, we denote $c(x)$ as the number of elementary cycles in $G(x)$. By Lemma 1, we are easily to compute $c(x)$ using the following formula.

Lemma 2. *For each node $x \in DT(G)$, $c(x)$ can be computed recursively as follows:*

$$c(x)=\begin{cases} 0, & \text{if } x \text{ is a leaf in } DT(G) \\ c(x_L)+c(x_R)+p(x_L)\cdot p(x_R), & \text{otherwise.} \end{cases}$$

Proof. Again, we consider only internal node. Note, by definition, $c(x_L)$ and $c(x_R)$ are the numbers of elementary cycles that contain only the vertices of $G(x_L)$ and of $G(x_R)$, respectively. Thus, we need to compute the number of elementary cycles that contain vertices from both $G(x_L)$ and $G(x_R)$. Let s_L and s_R be the sources, respectively, in the subgraphs $G(x_L)$ and $G(x_R)$. For any pair of vertices $u \in G(x_L)$ and $w \in G(x_R)$, there are $p(G(x_L),u)$ distinct elementary paths from s_L to u in $G(x_L)$ and there are $p(G(x_R),w)$ distinct elementary paths from s_R to w in $G(x_R)$. These paths together with the edges (u,s_R) and (w,s_L) produce $p(G(x_L),u)\cdot p(G(x_R),w)$ distinct elementary cycles in $G(x)$ (see Figure 4 for illustration). Thus, the total number of cycles in $G(x)$ is

$$\begin{aligned} &c(x) \\ &= c(x_L)+c(x_R)+\sum_{\substack{u\in G(x_L)\\ w\in G(x_R)}} p\left(G(x_L),u\right)\cdot p\left(G(x_R),w\right) \\ &= c(x_L)+c(x_R)+ \\ &\quad \left[\sum_{u\in G(x_L)} p\left(G(x_L),u\right)\right]\cdot\left[\sum_{w\in G(x_R)} p\left(G(x_R),w\right)\right] \\ &= c(x_L)+c(x_R)+p(x_L)\cdot p(x_R). \end{aligned}$$

□

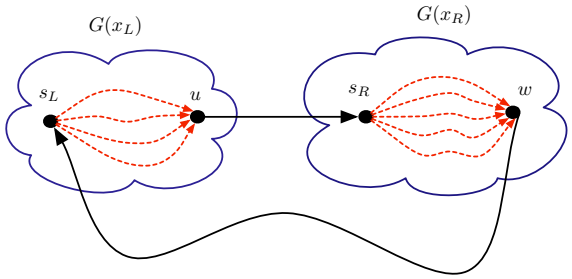


Figure 4: Illustration of Lemma 2.

For example, consider the MRF G shown in Figure 3 again. Recursively, applying the formulas given by Lemma 1 and Lemma 2, the number of elementary cycles of G can be computed in $DT(G)$ from bottom to up. Figure 5 shows the result.

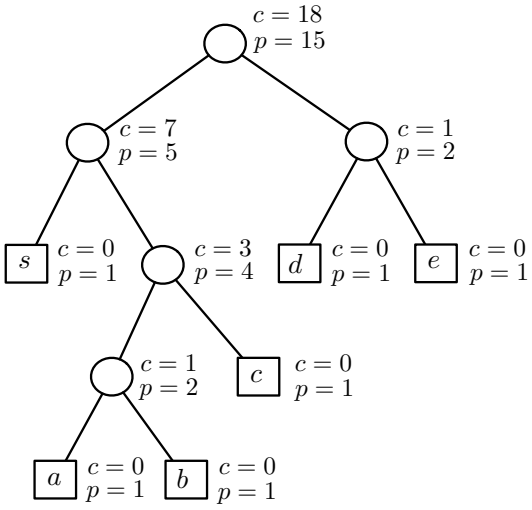


Figure 5: The calculation of the number of elementary paths and cycles in an MRF.

4. An enumeration algorithm

In this section, we shall propose an enumeration algorithm for generating all elementary cycles of an MRF. Indeed, we have a quite natural algorithm to do this according to the recursive formulas established in previous section. However, in general, the recursive process requires more space to preserve a set of node listings for representing the elementary paths in each subgraph $G(x)$. In the following, we provide another solution that uses only a linear space to generate all elementary cycles of an MRF. We take $DT(G)$ as input and assume that for each node $x \in DT(G)$, there are two fields $LL(x)$ and $RL(x)$ to record the leftmost leaf and the rightmost leaf in the subtree rooted at x, respectively. We initialize the both fields of leaves in $DT(G)$ to $0, 1, \ldots, n-1$ from left to right. Since every vertex of G corresponds to a leaf of $DT(G)$, this means that the vertices of G are numbered from 0 to $n-1$. For simplicity, $LL(x)$ and $RL(x)$ also denote the corresponding vertex of G. Note that $LL(x) = LL(x_L)$ for every internal node $x \in DT(G)$, where x_L is the left child of x. The following lemmas are related to the elementary cycles of MRFs.

Lemma 3. *For each internal node $x \in DT(G)$, let v_0 be the source of $G(x)$. If $C = (v_0, v_1, \ldots, v_k, v_0)$ is an elementary cycle in $G(x)$, then C contains only one back edge (v_k, v_0) (i.e., all other edges (v_i, v_{i+1}), $i = 0, 1, \ldots, k-1$, are forward edges).*

Proof. Since v_0 dominates v_k in $G(x)$, (v_k, v_0) is a back edge. Suppose, contrarily, that there exists $0 \leqslant i \leqslant k-1$ such that (v_i, v_{i+1}) is a back edge in $G(x)$. Then v_{i+1} dominates v_i in $G(x)$ (i.e., every path from v_0 to v_i in $G(x)$ must pass through v_{i+1}). Thus v_{i+1} is contained in the path $(v_0, v_1, \ldots, v_i)$, a contradiction to the assumption that C is an elementary cycle. □

Lemma 4. *For each internal node $x \in DT(G)$, let u, v be any two vertices such that u dominates v in $G(x)$. Suppose that C is an elementary cycle in $G(x)$ that contains the source of $G(x)$. If $v \in C$, then so is $u \in C$.*

Proof. Suppose $v \in C$. Then, C contains a path P from the source of $G(x)$ to v. Since u is a dominator of v in $G(x)$, P must pass through u, and therefore $u \in C$. □

We now introduce a two-pass tree-traversal procedure to compute $LL(x)$ and $RL(x)$ for every internal $x \in DT(G)$. These fields contain the local information about the dominance relation of G. Consider a node $x \in DT(G)$ with $LL(x) = i$. Then, i is the source of $G(x)$. Further, $LL(x)$ and $RL(x)$ also indicate the range (i.e., the lower and the upper) of vertices which can be dominated by i in $G(x)$. In addition, the procedure uses an auxiliary array Dom of size n to store the overall dominance of G. The assignment $Dom[i] = j$ means that the range of vertices dominated by i in G is between i and j.

Procedure DOMINANCE

1. $i \leftarrow 0$;
2. Apply the preorder to traverse $DT(G)$ and do the following:
   if the visited node x is an internal node then
   $LL(x) \leftarrow i$;
   else
   $i \leftarrow i + 1$;
3. Apply the postorder to traverse $DT(G)$ and do the following:
   if the visited node x is a leaf then
   $j \leftarrow RL(x)$;
   else
   $RL(x) \leftarrow j$;
   $Dom[LL(x)] \leftarrow RL(x)$;

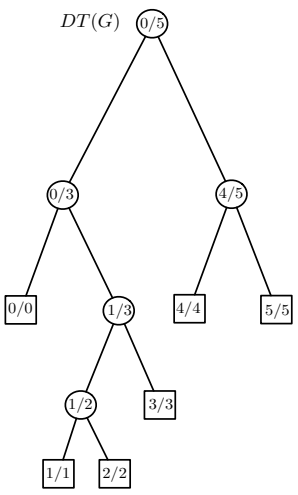
It is easy to construct the dominator tree of G using the field $LL(x)$ for nodes $x \in DT(G)$ in linear time. The constructing rule relies on the following lemma.

Lemma 5. *Let $x \in DT(G)$ be an internal node and x_R be the right child of x. Then $LL(x) = idom(LL(x_R))$.*

Proof. Suppose $LL(x) = i$ and $LL(x_R) = j$. By Theorem 1, $G(x) = G(x_L) \otimes G(x_R)$. Thus $j \in G(x)$. Let s be the source of G. Since i is the source of $G(x)$, every path from s to a vertex $v \in G(x)$ must pass through i. This shows that i dominates j in G. If i is not the immediate dominator of j, then there is a vertex $k \neq i$ such that $k = idom(j)$ in G. In this case, k must be the source of some subgraph $G(y)$, where y is a descendant of x and is also an ascendent of x_R in $DT(G)$. This leads a contradiction. □

Now to demonstrate the usage of the procedure, let's consider the decomposition tree $DT(G)$ shown in Figure 3. Figure 6(a) shows the values of $LL(x)$ and $RL(x)$ for nodes $x \in DT(G)$. The variance of Dom during the execution of DOMINANCE is shown in Figure 6(b). By Lemma 5, it can be seen from Figure 6(a) that 0 is the immediate dominator of 1 and 4, 1 is the immediate dominator of 2 and 3, and 4 is the immediate dominator of 5. Therefore, the dominator tree of G can be shown as in Figure 6(c).

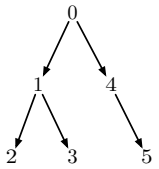
In what follows, we sketch our enumeration algorithm for generating all elementary cycles of an MRF and then give the details.



(a)

i	0	1	2	3	4	5
$Dom[i]$	0	1	2	3	4	5
	3	2			5	
	5	3				

(b)



(c)

Figure 6: (a) $LL(x)/RL(x)$ for nodes $x \in DT(G)$; (b) The contents of Dom and their variance; (c) The dominator tree of G.

Algorithm ENUMERATING-CYCLES

1. Perform DOMINANCE to compute $LL(x), RL(x)$ for all $x \in DT(G)$ and $Dom[i]$, $i = 1, \ldots, n$;
2. Apply a tree-traversal to visit the nodes of $DT(G)$ and for each internal node x, it produces the set $\mathscr{C}(x)$ containing all elementary cycles of $G(x)$ that pass through the source of $G(x_L)$ and the source of $G(x_R)$, simultaneously.

Obviously, $C \in \mathscr{C}(x)$ if and only if C is an elementary cycle of G such that the following two conditions are satisfied: (1) every vertex $v \in C$ is within the range between $LL(x)$ and $RL(x)$ and (2) C contains two particular vertices $LL(x_L)$ and $LL(x_R)$. To represent a cycle $C \in \mathscr{C}(x)$ starting from the source of $G(x)$ (i.e., $LL(x)$), we omit the unique back edge because it has been shown that the back edge is pointed to the source of $G(x)$ by Lemma 3. Also, since the leaves of $DT(G)$ are numbered $0, 1, \ldots, n-1$ from left to right, every forward edge in C has label from a smaller vertex to a larger vertex. Thus, an elementary cycle $C \in \mathscr{C}(x)$ can be viewed as an increasing sequence starting from $LL(x)$ and growing within the range between $LL(x)$ and $RL(x)$. As a result, a simple approach based on constructing of all advisable increasing sequences can be used to generate all elementary cycles of $\mathscr{C}(x)$. Although all increasing sequence within the range between $LL(x)$ and $RL(x)$ are candidates, some of them are unadvisable. We use the technique of constructing power sets to produce the desired candidates. For eliminating the unadvisable sequences, it can be achieved by using the information about dominance relation. Many textbooks contain implementations of generating the entire power set of n values. The primary expositions on this topic include [6, 8].

Let $I(x) = \{i \in \mathbb{N} \,|\, LL(x) \leqslant i \leqslant RL(x)\}$ and $J(x) = I(x) \backslash \{LL(x_L), LL(x_R)\}$, where $x \in DT(G)$ is an internal node. The *semi-power set* of $J(x)$, denoted by $\mathscr{P}(x)$, is a collection of all subsets of $J(x)$ such that $\hat{J} \in \mathscr{P}(x)$ if and only if whenever $u, v \in J(x)$ and u dominates v in $G(x)$, we have $u \in \hat{J}$ implies $v \in \hat{J}$. Now, by lemma 4, every elementary cycle of $\mathscr{C}(x)$ can be considered as an increasing sequence of integers obtained from $I(x)$ by removing the elements of some set $\hat{J} \in \mathscr{P}(x)$.

For example, we suppose that the algorithm ENUMERATING-CYCLES takes the decomposition tree in Figure 6(a) as input and consider x to be the root of the tree. Then $LL(x) = LL(x_L) = 0$, $RL(x) = 5$, and $LL(x_R) = 4$. From the above definitions, we have

$$I(x) = \{0, 1, 2, 3, 4, 5\},$$

$$J(x) = \{1, 2, 3, 5\},$$

and

$$\begin{aligned} \mathscr{P}(x) \;=\; & \{\emptyset, \{5\}, \{3\}, \{3, 5\}, \{2\}, \{2, 5\}, \{2, 3\}, \\ & \{2, 3, 5\}, \{1, 2, 3\}, \{1, 2, 3, 5\}\}. \end{aligned}$$

It should be noted that since 1 dominates 2 and 3 in G, if 1 presents in a set $\hat{J}$, so are $2, 3 \in \hat{J}$. Thus, all elementary cycles of $\mathscr{C}(x)$ are listed as follows:

$$\begin{aligned} & (0, 1, 2, 3, 4, 5), (0, 1, 2, 3, 4), (0, 1, 2, 4, 5), \\ & (0, 1, 2, 4), (0, 1, 3, 4, 5), (0, 1, 3, 4), \\ & (0, 1, 4, 5), (0, 1, 4), (0, 4, 5), (0, 4). \end{aligned}$$

In real implementation, we generate semi-power set with *inclusion property*. That is, if $\hat{J}, \tilde{J} \in \mathscr{P}(x)$ and $\hat{J} \subset \tilde{J}$, then $\hat{J}$ must be generated before $\tilde{J}$. In fact, most of algorithms proposed in the literature enumerate power sets with lexicographic order,

which imply that they possess the inclusion property. Suppose $J(x) = \{j_1, j_2, \ldots, j_k\}$ with $j_i \leqslant j_{i+1}$ for all $i = 1, \ldots, k-1$ and let $\hat{J} = \{j_{i_1}, j_{i_2}, \ldots, j_{i_l}\}$ be any subset of $J(x)$ such that $j_{i_m} \leqslant j_{i_{m+1}}$ for all $m = 1, \ldots, l-1$. When $\hat{J}$ is to be generated by the algorithm, it invokes two processes, one is the *set-extending* and the other is the *membership-checking*. The former extends $\hat{J}$ into a larger set, if possible, such that the result is a candidate of the members in $\mathscr{P}(x)$. The latter decides whether the candidate is indeed a set in $\mathscr{P}(x)$. The above two processes can be implemented together and rely on the dominance relation for checking. For efficiency, we don't need to explore the dominance relation between any two elements of $J(x)$. We only need to inspect the dominance relation for pairs of successive vertices in increasing order. The following is the principle: if there exists $m \in \{1, \ldots, l-1\}$ such that $Dom[j_{i_m}] > j_{i_{m+1}}$, then $\hat{J} \notin \mathscr{P}(x)$. Otherwise, we extend $\hat{J}$ by adding the vertices between j_{i_m} and $Dom[j_{i_m}]$ for each $m \in \{1, \ldots, l-1\}$.

For instance, consider $\hat{J} = \{1, 5\}$ and $\tilde{J} = \{1, 2, 5\}$ in the above example. By inclusion property, we generate $\hat{J}$ before $\tilde{J}$. When $\hat{J}$ is to be generated by the algorithm, since $Dom[1] = 3 < 5$, vertices 2 and 3 are forced to join the set. Thus, $\{1, 2, 3, 5\} \in \mathscr{P}(x)$. Later on, when $\tilde{J}$ is to be generated by the algorithm, since $Dom[1] = 3 > 2$, this implies $\tilde{J} \notin \mathscr{P}(x)$.

The correctness of the algorithm directly follows from Lemma 4. We now illustrate why that Enumerating-Cycles is efficient in space. In fact, we don't need to use extra space to store $I(x)$, $J(x)$ and its semi-power set for every internal node $x \in DT(G)$. Oppositely, a global array can be repeatedly used to mark which elements are contained in the presenting cycle for each internal node $x \in DT(G)$. Recall that $LL(x)$ and $RL(x)$ indicate the boundaries of vertices in the presenting cycle. For each set $\hat{J} \in \mathscr{P}(x)$, only a piece size of elements in the array between the boundaries need to change their statuses for listing the cycle. This shows that it takes time proportional to the length of a cycle for each enumerating. Therefore, we summarize our result as the following theorem.

Theorem 2. *Given a decomposition tree of an MRF G, Algorithm* Enumerating-Cycles *can correctly output all elementary cycles of G using linear space.*

5. Conclusions

In this paper, we give a recursive formula for counting the number of elementary cycles of an MRF when its corresponding decomposition tree is given. Moreover, based on dominance relation of vertices in the graph and the technique of generating power set of a set, we design an enumeration algorithm using linear space for listing all elementary cycles. Since explicit manipulation of power sets quickly is intractable due to their size, no further analysis on the time complexity are provided in this paper. According to Lemma 1 and Lemma 2, it is easy to check that the lower bound and the upper bound of the number of elementary cycles in MRFs with n vertices are $n(n-1)/2$ and $2^{n-1} - 1$, respectively. Two extreme instances realizing these bounds are the right-skew tree and the left-skew tree with n vertices. A continued work of this research is how to generalize the proposed enumeration algorithm on larger classes of flowgraphs. Another line will be devoted to develop efficient algorithms using the decomposition tree on MRFs.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, MA, 1986.

[2] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[3] S. Chen and D. R. Ryan, A comparison of three algorithms for finding fundamental cycles in a directed graph, *Networks* 11 (1981) 1–12.

[4] M. S. Hecht and J. D. Ullman, Flow graph reducibility, *SIAM J. Comput.* 1 (1972) 188–202.

[5] M. S. Hecht and J. D. Ullman, Characterization of reducible flow graphs, *J. Assoc. Comput. Mach.* 21 (1974) 367–375.

[6] A. Nijenhuis and H. Wilf, *Combinatorial Algorithms for Computers and Calculators*, Academic Press, Orlando, FL, 1978.

[7] R. C. Read and R. E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks* 5 (1975) 237–252.

[8] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[9] R. E. Tarjan, Enumeration of elementary circuits of a directed graph, *SIAM J. Comput.* 2 (1974) 211–216.

[10] J. C. Tiernan, An efficient search algorithm to find the elementary circuits of a graph, *Comm. ACM* 13 (1970) 722–726.

[11] O. Vernet and L. Markenzon, Characterizations and properties of maximal reducible flowgraphs, *Congr. Numer.* 139 (1999) 9–20.

[12] O. Vernet and L. Markenzon, Solving problems for maximal reducible flowgraphs, *Discrete Appl. Math.* 136 (2004) 341–348.

[13] H. Weinblatt, A new search algorithm for finding the simple cycles of a finite directed graph, *J. ACM* 19 (1973) 43–56.

[14] J. T. Welch, A mechanical analysis of the cyclic structure of undirected linear graphs, *J. ACM* 13 (1966) 205–210.