SIAM J. COMPUT.
Vol. 2, No. 3, September 1973

ENUMERATION OF THE ELEMENTARY CIRCUITS OF A DIRECTED GRAPH*

ROBERT TARJAN†

Abstract. An algorithm to enumerate all the elementary circuits of a directed graph is presented. The algorithm is based on a backtracking procedure of Tiernan, but uses a lookahead and labeling technique to avoid unnecessary work. It has a time bound of $O((V \cdot E)(C + 1))$ when applied to a graph with V vertices, E edges, and C elementary circuits.

Key words. algorithm, backtracking, circuit, cycle, digraph, graph.

There is a class of problems in which the goal is the enumeration of a set of objects associated with a given graph. Examples include enumeration of the elementary circuits, enumeration of the spanning trees, and enumeration of the cliques of a given graph. For each of these three problems, graphs with V vertices may be constructed which have 2^V or more objects to be enumerated. Thus, in general, algorithms to solve these problems must have a running time exponential in the size of the graph. However, in practical applications, the number of objects to be enumerated is usually a much smaller function of the size of the graph. Thus it would be useful to find enumeration algorithms with running times polynomial in the number of objects generated. Presented here is an algorithm for enumerating elementary circuits whose running time is a small-degree polynomial function of the size of its input and output.

A (*directed*) *graph* $G = (\mathscr{V}, \mathscr{E})$ consists of a set of *vertices* $\mathscr{V}$ and a set of ordered pairs of vertices $\mathscr{E}$, called the *edges* of G. If (v, w) is an edge of G, vertices v and w are said to be *adjacent*. A *path* in a graph is a sequence of edges (v_1, v_2), $(v_2, v_3), \cdots, (v_{n-1}, v_n)$, such that the terminal vertex of an edge in the sequence is the initial vertex of the next edge. A path may be denoted by the sequence of vertices on it. An *elementary path* contains no vertex twice. An *elementary circuit* is an elementary path with the exception that its first and last vertices are identical. Two elementary circuits whose edge sequences are cyclic permutations of each other are regarded as identical. For simplicity we shall assume that a graph contains no self-loops (edges of the form (v, v)) and no multiple edges.

We wish to enumerate all the elementary circuits of a given graph. Tiernan [2] presents an algorithm for accomplishing this. His algorithm uses an essentially unconstrained backtracking procedure which explores elementary paths of the graph and checks to see if they are cycles. If the vertices of the graph are numbered from 1 to V, the algorithm will generate all elementary paths $p = (v_1, v_2, \cdots, v_k)$ with $v_1 < v_i$ for all $2 \leqq i \leqq k$, by starting from some vertex v_1, choosing an edge to traverse to some vertex $v_2 > v_1$, and continuing in this way. Whenever no new vertex can be reached, the procedure backs up one vertex and chooses a different edge to traverse. If v_1 is adjacent to v_k, the algorithm lists an elementary cycle $(v_1, v_2, \cdots, v_k, v_1)$. The algorithm enumerates each elementary cycle exactly once, since each such cycle contains a unique smallest vertex v_1 and thus corresponds

* Received by the editors October 17, 1972, and in revised form March 28, 1973.

† Department of Computer Science, Cornell University, Ithaca, New York 14850.

to a unique elementary path with starting vertex v_1. However, the algorithm has a worst-case running time exponential in the size of its output; it may explore many more elementary paths than are necessary. Consider the graph G in Fig. 1. It contains $3n + 1$ vertices, $5n$ edges and $2n$ elementary circuits. However, G contains 2^n elementary paths from vertex 1 to vertex $3n + 1$, all of which will be generated by Tiernan's algorithm. Thus the worst-case time bound of the algorithm is exponential in the number of elementary circuits, as well as exponential in the size of the graph.

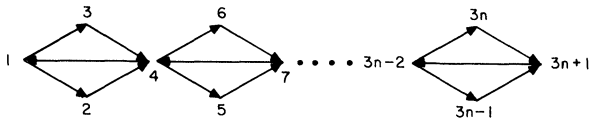

FIG. 1. *An example showing the inefficiency of Tiernan's algorithm*

Weinblatt [3] gives an algorithm for finding elementary circuits which is related to Tiernan's, but which requires substantially more bookkeeping. The algorithm has the property that it examines each edge of the graph exactly once. Given a graph G, we start from some vertex and choose an edge to traverse. We continue the search at each step by selecting an *unexplored* edge leading from the vertex most recently reached which still has unexplored edges. Eventually each edge of the graph will be traversed. Such a search is easy to program, because the set of old vertices with possibly unexplored edges may be stored on a stack. (See [1], for instance.) This sequence of vertices is an elementary path from the initial vertex to the vertex currently being examined. (Weinblatt calls it the TT, or "trial thread".) Whenever we traverse an edge leading to a vertex already on the stack, we have found a new elementary circuit, corresponding to a sequence of vertices on top of the stack. Whenever we traverse an edge leading to an old vertex which is *not* currently on the stack, some portion of the stack plus a sequence of subpaths from circuits already found may form a new elementary circuit. Weinblatt uses a recursive backtracking procedure to test combinations of subpaths from old circuits to see if they give new circuits in this way.

Although Weinblatt's algorithm is often much more efficient than Tiernan's, the recursive backtracking procedure requires exponential time in the worst case. For example, consider the graph in Fig. 2. It contains $3n + 2$ vertices, $5n + 3$ edges and $2n + 2$ elementary circuits. Suppose we start Weinblatt's algorithm by exploring the edge $(0, 1)$. Then the algorithm will generate all circuits of the form $(3i - 2, 3i + 1, 3i, 3i - 2)$ and $(3i - 2, 3i + 1, 3i - 1, 3i - 2)$ rapidly. Eventually the algorithm will traverse edge $(0, 3n + 1)$. Then Weinblatt's recursive procedure will attempt to find an elementary path back to vertex 0 by combining parts of old circuits. The recursive backtracking will require an exponential amount of time but will produce only one new circuit, i.e., $(0, 3n + 1, 0)$. Thus Weinblatt's algorithm does not have a running time polynomial in the number of circuits.

However, it *is* possible to construct a polynomial-time algorithm for the circuit enumeration problem. Such an algorithm uses Tiernan's backtracking

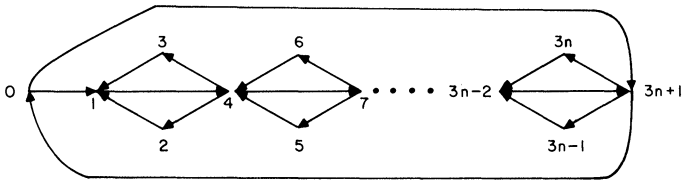

Fig. 2. *An example showing the inefficiency of Weinblatt's algorithm*

procedure restricted so that only fruitful paths are explored. The circuit enumeration algorithm is presented below in ALGOL-like notation. The algorithm assumes that the vertices of the graph are numbered from 1 to V, and that the graph is represented by a set of adjacency lists, one for each vertex. The adjacency list $A(v)$ of vertex v contains all vertices w such that (v, w) is an edge of the graph. The point stack used in the algorithm denotes the elementary path p currently being considered; the elementary path has start vertex s. Every vertex v on such a path must satisfy $v \geqq s$. A vertex v becomes *marked* when it lies on the current elementary path p. As long as v lies on the current elementary path or it is known that every path leading from v to s intersects p at a point other than s, v stays marked.

For each vertex s, the algorithm generates elementary paths which start at s and contain no vertex smaller than s. Once a vertex v has been used on a path, it can only be used to extend a new path when it has been deleted from the point stack *and* when it becomes unmarked. A vertex v becomes unmarked when it might lie on a simple circuit which is an extension of the current elementary path. Whenever the last vertex on an elementary path is adjacent to the start vertex s, the elementary path corresponds to an elementary circuit which is enumerated. The marking procedure is the key to avoiding the unnecessary searches which may occur when using Weinblatt's algorithm or Tiernan's original algorithm.

ALGORITHM.

```
procedure circuit enumeration;
    begin
        procedure BACKTRACK (integer value v, logical result f);
            begin
                logical g;
                f := false;
                place v on point stack;
                mark (v) := true:
                place v on marked stack;
                for w ∈ A(v) do
                    if w < s then delete w from A(v)
                    else if w = s then
                        begin
                            output circuit from s to v to s given by point
                            stack;
                            f := true;
```

```
                    end
                else if ¬ mark (w) then
                    begin
                        BACKTRACK (w, g);
                        f := f ∨ g;
                    end;
            comment f = true if an elementary circuit containing the
                partial path on the stack has been found;
            if f = true then
                begin
                a: while top of marked stack ≠ v do
                        begin
                            u := top of marked stack;
                            delete u from marked stack;
                            mark (u) := false;
                        end;
                    delete v from marked stack;
                    mark (v) := false;
                end;
            delete v from point stack;
        end;
    integer n;
    for i := 1 until V do mark (i) := false;
    for s := 1 until V do
        begin
        b: BACKTRACK (s, flag);
            while marked stack not empty do
                begin
                    u := top of marked stack
                    mark (u) := false;
                    delete u from marked stack;
                end;
        end;
end;
```

LEMMA 1. *Let $c = (v_1, v_2, \cdots, v_n, v_1)$ be an elementary circuit in a graph G, satisfying $v_i > v_1$ for $2 \leqq i \leqq n$. Let the circuit enumeration procedure be applied to G. Consider the execution of statement b*: BACKTRACK (s, flag), *with $s = v_1$. During the execution of this statement (which may involve recursive calls on* BACKTRACK), *we have: for all $1 \leqq k \leqq n$, if v_k is marked, then for some $j \geqq k$, v_j is on the point stack.*

Proof. We prove the lemma by induction on k, with k decreasing. Let $k = n$. Suppose v_n becomes marked. Then it is added to the point stack at the same time. Before v_n is removed from the point stack, variable f becomes true, since (v_n, v_1) is an edge of the graph. Thus v_n will become unmarked when it is deleted from the stack, and the lemma is true for $k = n$.

Let the lemma be true for v_i with $i > k$. We prove the lemma for v_k. Suppose v_k becomes marked. Then v_k is placed on the point stack. There are two cases:

(a) Vertex v_k is placed on top of some v_j on the point stack with $j > k$. Then v_j is also underneath v_k on the marked stack. Thus if v_k is marked, v_j is also marked. But if v_j is marked, there is some v_l with $l \geqq j$ on the point stack by the induction hypothesis, and since $l \geqq j > k$, the lemma holds for v_k.

(b) Vertex v_k is not placed on top of any v_j on the point stack with $j > k$. Then when the edge (v_k, v_{k+1}) is examined, v_{k+1} is unmarked by the induction hypothesis, and will be added to the stack. Subsequently $v_{k+2}, v_{k+3}, \cdots, v_n$ will be added to the stack, and an elementary circuit containing v_k will be found. A flag is set true to note this discovery. The flag filters up through recursive returns from BACKTRACK, and v_k will be unmarked by step a when v_k is removed from the point stack. Thus the lemma holds for v_k.

By induction, the lemma holds in general.

LEMMA 2. *The circuit enumeration algorithm lists each elementary circuit of a given graph exactly once.*

Proof. The starting vertex of any elementary path p generated by the algorithm is the lowest numbered vertex on the path p. Since the algorithm generates an elementary path at most once, and since an elementary circuit has only one lowest numbered vertex, each elementary circuit is generated at most once.

Let $c = (v_1, v_2, \cdots, v_n, v_1)$ be an elementary circuit. Let the circuit enumeration procedure be applied to G. Consider the execution of step b: BACKTRACK (s, flag) with $s = v_1$. If $(v_1, \cdots, v_k)$ is on the point stack for any $1 \leqq k \leqq n$, then by Lemma 1, v_{k+1} must be unmarked, and v_{k+1} will be added to the point stack on top of v_k when edge (v_k, v_{k+1}) is examined. By induction, $(v_1, v_2, \cdots, v_n)$ will eventually be on the point stack, and the algorithm will enumerate the circuit c. Thus each elementary circuit is generated at least once.

LEMMA 3. *If G is a graph with V vertices and E edges, applying the circuit enumeration algorithm to G requires $O(V + E + S)$ space, where S is the sum of the lengths of all the elementary circuits, and $O(VE(C + 1))$ time, where C is the number of elementary circuits.*

Proof. The space bound is obvious; storage of the graph's adjacency lists requires $O(V + E)$ space, storage for the algorithm's data structures requires $O(V)$ space, and storage for the output requires $O(S)$ space. If we do not want to store all the elementary circuits after they are generated the algorithm requires only $O(V + E)$ space.

The time bound follows from the following observation: after a circuit c is enumerated, but before another circuit with the same start vertex is enumerated, any vertex can become unmarked at most V times (at most once for each vertex on the point stack when c is enumerated). Thus after c is enumerated but before another circuit with the same start vertex is enumerated, any edge can be explored at most V times. After the start vertex has changed but before any new circuits are enumerated, any edge can be explored only once, since no vertices become unmarked during this time. Thus the algorithm requires $O(E)$ time for each start vertex plus $O(VE)$ time for each circuit, for a total time of $O(VE(C + 1))$. In terms of V, E and S, the time bound is $O(E(S + V))$.

The variation of Tiernan's algorithm presented here has a running time

polynomial in the size of the input and output, a fact not true in general for either Tiernan's original circuit generation algorithm or for Weinblatt's algorithm. However, the number of circuits may be exponential in the number of vertices. On graphs with very large numbers of circuits, all three algorithms run in time exponential in the size of the problem graph. On graphs with simple looping structure, Weinblatt's algorithm may run faster than the one presented here by a factor of at most V, but the new algorithm may be speeded up in this case if a little preprocessing is done. Tiernan's original algorithm is slightly simpler to program than the version presented here, but by avoiding the labeling process it may perform many unnecessary searches. The new version has a provably reasonable time bound on all types of graphs.

The new algorithm was implemented in ALGOL W, the Stanford University version of ALGOL (see [4]), and run on a variety of sample graphs using an IBM 360/65. The program counted 125,664 circuits in a nine-vertex complete graph in 101.2 seconds. The running time of the new algorithm was faster than that claimed by either Tiernan or Weinblatt, although they used slower computers (B-5500 and an IBM 7094, respectively). The new algorithm is certainly competitive in practice. It is still an open question whether a circuit enumeration algorithm exists whose time bound is *linear* in the size of its input and output.

Acknowledgments. The author wishes to thank the referees and Professors Lloyd Fosdick and Leon Osterweil of the University of Colorado for their discovery of errors in the original formulation of the algorithm, and Professor John Hopcroft of Cornell University, who helped verify the corrections.

REFERENCES

[1] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

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

[3] H. WEINBLATT, *A new search algorithm for finding the simple cycles of a finite directed graph*, J. Assoc. Comput. Mach., 19 (1972), pp. 43–56.

[4] R. L. SITES, *Algol W Reference Manual*, Tech. Rep. STAN-CS-71-230, Computer Science Dept., Stanford University, Stanford, California, August, 1971.