ENUMERATION OF THE ELEMENTARY CIRCUITS OF A DIRECTED GRAPH

Robert Tarjan

TR 72-145

September 1972

Department of Computer Science
Cornell University
Ithaca, N.Y. 14850

Enumeration of the Elementary Circuits of a Directed Graph

Robert Tarjan
Computer Science Department
Cornell University

Abstract:

An algorithm to enumerate all the elementary circuits of a directed graph is presented. The algorithm uses backtracking with lookahead to avoid unnecessary work, and it has a time bound of $O((V + E)(C+1))$ when applied to a graph with V vertices, E edges, and C elementary circuits.

Keywords: Algorithm, circuit, cycle, graph

Many applications require the enumeration of a certain set of objects associated with a given graph. In some cases the number of objects may grow exponentially with the number of vertices in the graph; thus there are no algorithms with time time bounds polynomial in the size of the graph for solving such problems. Examples include enumerating the elementary circuits, the spanning trees, or the cliques of a given graph. However, it may be possible to find algorithms with time bounds polynomial (and hopefully linear) in the number of objects generated. Presented here is an algorithm for enumerating elementary circuits in a graph which is "good" in this sense.

A (directed) graph $G=(\mathcal{V},\mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of ordered pairs of vertices $\mathcal{E}$, 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. 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 accom-

plishing 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 $p=(v_1,v_2,\ldots,v_k)$ with $v_1<v_i$ for all $2\leq i\leq k$, 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 $(v_1,v_2,\ldots v_k,v_1)$. The algorithm enumerates each elementary cycle exactly once, since each such cycle contains a unique smallest vertex v_1 and thus corresponds to a unique elementary path with starting vertex v_1. However, the algorithm is not a "good" one, because it explores many more elementary paths than are necessary. Consider the graph G in Figure 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.

Weinblatt [3] gives an algorithm for finding elementary circuits which is related to Tiernan's, but which requires substantially more bookkeeping. Weinblatt uses a depth-first search to explore the graph. A general description of the depth-first search technique appears in [1]. Given a graph G, we start from some vertex and choose an edge to follow. This edge leads to a

new vertex. We continue in this way; at each step we select an unexplored edge leading from a vertex already reached and we traverse this edge. The edge leads to some vertex, either new or already reached. Whenever we run out of edges leading from old vertices we choose some unreached vertex, if any, and begin a new exploration from this point. Eventually we will traverse all the edges of G, each exactly once. (A backtracking procedure such as Tiernan's may traverse each edge of a graph many times.) Such a process is called a search of G. Suppose we use the following choice rule to select the next edge to traverse: always choose an edge emanating from the vertex most recently reached which still has unexplored edges. A search which uses this rule is a depth-first search.

A depth-first search is easy to program, because the set of old vertices with possibly unexplored edges may be stored on a stack. 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 Figure 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 cycles. The recursive backtracking will require an exponential amount of time but will produce only one new

cycle; namely (0, 3n+1, 0). Thus Weinblatt's algorithm is not "good" in the sense we desire.

However, it is possible to construct a "good" algorithm for the circuit enumeration problem. Such an algorithm uses a backtracking 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 $v \geq s$. A vertex v is marked if it is on the elementary path p or if every path leading from v to s intersects p at a point other than s.

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 in a path, it can only be used in a new path when it has been deleted from the point stack and when it becomes unmarked. A vertex v becomes unmarked when a path from v to s which does not intersect the current elementary path other than at s is found. 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.

```
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 v g;
            end;
          comment f=true if an elementary circuit continuing the
            partial path on the stack has been found;
          if f=true then
            begin
              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
      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 $c = (v_1, v_2, \ldots, v_n, v_1)$ be an elementary circuit in a graph G. Consider applying BACKTRACK(v_1) to G. For all $1 \leq k \leq n$, v_k will be unmarked unless, for some $j \geq k$, v_j is on the stack.

Proof: Vertex v_n is unmarked unless it is on the stack, since (v_n, v_1) is an edge of the graph and an elementary circuit containing v_n will be found each time v_n is placed on the stack, causing v_n to be unmarked whenever it is removed from the stack.

Let the Lemma be true for v_i with $i>k$. Suppose v_k is placed on the stack. If v_k is placed on top of some v_j, $j>k$, then v_k will become unmarked before v_j becomes unmarked (since v_k appears on top of v_j in the marked stack). While v_j is marked, there is some v_ℓ with $\ell \geq j$ on the stack by the induction hypothesis, and since $\ell \geq j \geq k$ the Lemma holds for v_k.

On the other hand if v_k is not placed on top of any v_j, $j>k$, then by the induction hypothesis all the v_j, $j>k$, are unmarked. Thus when the edge (v_k, v_{k+1}) is examined, v_{k+1} will be unmarked and will be added to the stack. Subsequently $v_{k+2}, v_{k+3}, \ldots, v_n$ will be added to the stack and an elementary circuit containing v_k will be found. Thus v_k will be unmarked when it is removed from the point stack and 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 in 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 $c = (v_1, v_2, \ldots, v_n, v_1)$ be an elementary circuit. Consider the application of BACKTRACK(v_1) to a graph. If $(v_1, \ldots, v_k)$ is on the point stack for any $1 \leq k \leq n$, 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 $(v_1, v_2, \ldots, v_n)$ 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((V+E)(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 observation that after a vertex is marked, it cannot become unmarked until a new circuit is generated. Thus only O(V+E) time elapses between the generation of two circuits, and the total circuit generation time is O((V+E)(C+1)). This bound is tight for the algorithm presented here, as one may see by constructing suitable example graphs.

A "good" algorithm for the enumeration of all the elementary circuits of a directed graph has been presented. The algorithm is not only good in the sense that its theoretical time and space bounds are polynomial (in fact, bilinear) in the size of its input and output, but the algorithm is simple to understand, easy to program, and superior to other algorithms in the literature. The algorithm uses backtracking with look-ahead, an idea which is easily adaptable to other enumeration problems. It is an open question whether a circuit enumeration algorithm exists whose time bound is linear in the size of its input and output.

References:

[1] Tarjan, R., "Depth-first search and linear graph algorithms", SIAM J. Comput., Vol. 1, No.2 (June, 1972), 146-160.

[2] Tiernan, J.C., "An efficient search algorithm to find the elementary circuits of a graph", CACM, Vol. 13, No. 12 (Dec., 1970), 722-726.

[3] Weinblatt, H., "A new search algorithm for finding the simple cycles of a finite directed graph," JACM, Vol 19, No. 1 (Jan 1972), 43-56.

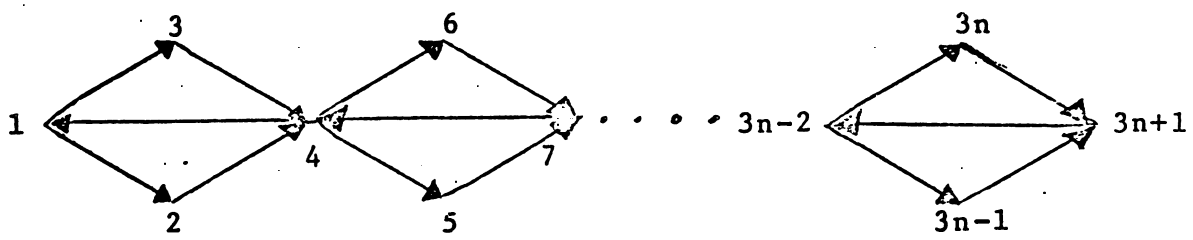

Figure 1: An example showing the inefficiency of Tiernan's algorithm.
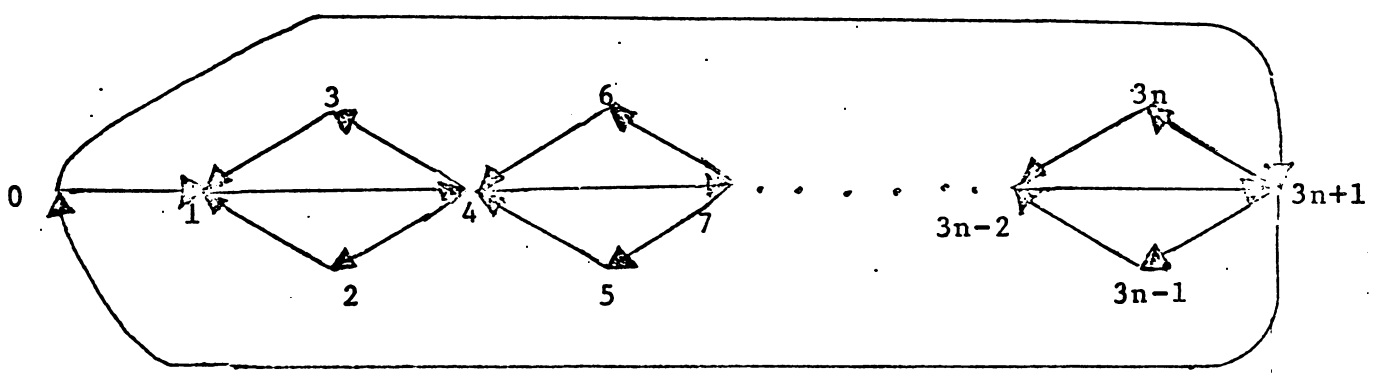
Figure 2: An example showing the inefficiency of Weinblatt's algorithm.