BIT 16 (1976), 192–204

A SEARCH STRATEGY FOR THE ELEMENTARY CYCLES OF A DIRECTED GRAPH

JAYME L. SZWARCFITER* and PETER E. LAUER

Abstract.

The most successful known algorithms enumerating the elementary cycles of a directed graph are based on a backtracking strategy. Such existing algorithms are discussed and a new backtracking algorithm is proposed which is bounded by $O(N+M(C+1))$ time, for a directed graph with N vertices, M edges and C elementary cycles.

1. Introduction.

Some problems, such as determining whether a graph has certain properties, or constructing a set of objects related to the graph admit of algorithmic solutions which have a time bound linear in the size of the graph. These include the problems of finding: the strongly connected components of a directed graph (Tarjan [18]), the biconnected components of a graph (Tarjan [18]), the graph from its given line graph (Roussopoulos [14], Lehot [9]), partitions of a graph into simple paths (Hopcroft and Tarjan [5]). Testing planarity of a graph can be performed in a time just proportional to the number of its vertices (Hopcroft and Tarjan [6]). Clearly, any algorithm for listing a set of objects related to the graph *must* be at least proportional to the total number of such objects. If this number grows exponentially with the size of the graph, the algorithm has at least an exponential running time. For such problems, a given algorithm may be additionally characterized by introducing a time bound per object obtained, and two algorithms can be compared according to their bounds per object. The problem of finding all elementary cycles of a directed graph falls into this category. Among the great number of cycle algorithms surveyed by Prabhaker and Deo [10], the algorithm by Johnson [7], presents the best time bound, namely a linear bound in the size of the graph, per cycle. This algorithm was devised by imposing further constraints on the backtracking performed by an already constrained backtracking algorithm [19].

* Research supported by the Conselho National de Desenvolvimento Científico e Tecnológico – CNPq – Brasil.

Received July 17, 1975. Revised February 7, 1976.

The present paper proposes a cycle finding algorithm that has a similar (worst case) time bound as [7]. However, while maintaining all the constraints of [7], we are proposing new strategies that represent further restrictions to the backtracking.

The graph definitions presently adopted are those commonly found in the literature. Refer for instance to [18]. By N, M, C respectively, we denote the number of vertices, edges and elementary cycles of a digraph $D(V,E)$, with V the set of vertices and E the set of edges. We assume that $V=\{1,\ldots N\}$.

2. Elementary cycles in directed graphs.

Tiernan [20] finds all elementary paths $v_1,\ldots,v_k$, $v_1<v_i$, $1<i\leqq k$ and $1\leqq k\leqq N$. If $(v_k,v_1)\in E$ then the cycle $v_1,\ldots,v_k,v_1$ is enumerated. This strategy corresponds to an essentially unconstrained backtracking and was also presented by Roberts and Flores [13] and Berztiss [1]. Floyd [4] has described a non-deterministic version of this algorithm. Weinblatt [21] also searches for elementary paths, but proposes to improve execution time by storing cycles already found and constructing new ones from these. In [19] Tarjan gives examples illustrating that the algorithms [20] and [21] may take exponential time in the number of cycles enumerated. Lauer [8] discusses the generalization of Tiernan's algorithm to different representations of digraphs, improves storage requirements and proposes alternate proofs. Another backtracking algorithm presented by Berztiss [2] has been shown by Prabhaker and Deo [10] also to have a time bound exponential in the number of cycles. The algorithm by Syslo [15, 16] is also based on a backtracking strategy and constitutes a variation of Tiernan's method.

Tarjan's algorithm [19] is based on Tiernan's depth-first method. It makes use of two stacks, the *point stack* for storing the path currently being examined and a *mark stack*, as well as a boolean vector called *mark vector*. The mark stack is used as a set of pointers to the mark vector. Whenever a new cycle is found, all vertices in the current point stack will eventually be unmarked when popped from this stack.

If no cycle is found involving a vertex, it will be deleted from the point stack, but continue to be marked. Some of the unnecessary work done by Tiernan is avoided by the condition that a vertex can be added to the point stack only if it is found unmarked. However, we mention two points where this algorithm still does unnecessary work. First, whenever a vertex v is going to be unmarked because a cycle involving it was found, *all* the vertices that are above v in the mark stack will also be

unmarked, even if some of them are involved in no cycle. Second, Tarjan follows Tiernan's principle of only searching for elementary cycles $v_j, \ldots, v_k$ with $v_j < v_i$, $1 < i \leqq k$, where v_j is the vertex at the bottom of the stack, called *start vertex*. The inefficiency involved in this is discussed in section 5. The algorithm [19] is bounded by $O(NM(C+1))$ time and $O(N+M)$ space.

Another method was developed by Ehrenfeucht, Fosdick and Osterweil [3] which includes both breadth-first and depth-first search, and makes use of an additional phase for collecting information about the digraph. This preprocessing requires $O(N^3)$ time and the actual process enumeration of the elementary cycles is bounded by $O(NM)$ time per cycle.

An algorithm with the time bound of $O(N+M(C+1))$ has been presented by Read and Tarjan [12]. It should be noted that a first version of the present paper contained a counter-example to a previous formulation of the Read and Tarjan algorithm [11], showing that the strategy described in [11] has a time bound greater than $O(N+M(C+1))$.

The algorithm by Johnson [7] also employs the technique of constructing elementary paths from a start vertex, in a stack. For each strongly connected component, the least vertex of the component becomes the start vertex. Subsequently, a new maximal strongly connected partial subdigraph is obtained, which does not contain that vertex. The least vertex of this partial subdigraph becomes the new start vertex and so on. For each start vertex s, a recursive backtracking procedure is invoked and its computation is similar to that of Tarjan's algorithm, except for the marking system, which was considerably enhanced. A vertex v is marked each time it enters the stack. Upon leaving the stack, if an elementary cycle was found involving v and the start vertex s, then v is unmarked. Otherwise, it remains marked until another vertex u is popped from the stack and such that an elementary cycle existed involving u and s, and there exists a path from v to u consisting of vertices that are marked and not in the stack. Johnson implements this strategy efficiently using a scheme of lists B, one list $B(v)$ per vertex v. At any given moment, $B(v)$ contains those vertices u such that $(u,v) \in E$ and u is marked and not in the stack. The actual unmarking is performed by a procedure $UNBLOCK(v)$ which will recursively call $UNBLOCK(u)$, if $u \in B(v)$. This algorithm is bounded by $O(N+(C+1)M)$ time and $O(N+M)$ space. Further remarks concerning this method and comparisons with the proposed algorithm can be found in section 5.

3. The Proposed Algorithm.

Our algorithm also uses a recursive backtracking procedure but a more efficient system for detecting elementary cycles. This detection occurs as soon as the elementary cycle is generated anywhere in the current path under examination. This path is kept in a stack (Tarjan's point stack). The boolean vector is retained but not the mark stack. Instead, we have utilised and slightly modified Johnson's marking system using one list $B(v)$, per vertex v. A vertex u is inserted in list $B(v)$ if $(u,v) \in E$ and the exploration of edge (u,v) has not led to a new elementary cycle. In addition to these structures, we use a *position* vector and a boolean *reach* vector. If a vertex v is the jth vertex from the bottom of the stack, then position $(v)=j$; when v is deleted from the stack then position $(v)=N+1$. If a vertex v has not yet left the stack for the first time, then reach$(v)=$ **false**, otherwise reach$(v)=$ **true**. A vertex v is marked when it enters the stack, and the mark is kept at least as long as this vertex remains in the stack. Upon leaving the stack, v is unmarked only if a new elementary cycle was found with v but not necessarily with the vertex at the bottom of the stack (start vertex). If v leaves the stack with the mark on, then it will be unmarked when a vertex z_1 is popped from the stack in such a way that a new elementary cycle was found with z_1, and there exists a path $z_k, z_{k-1}, \ldots, z_1$, $(z_k = v)$ such that $z_{i+1} \in B(z_i)$, $k < i \leqq 1$, at that time.

The digraph is represented by a set of adjacency lists with one list $A(v)$ per vertex v. A pre-processing is performed to find the strongly connected components of the digraph, using the method described in [18]. For each strongly connected component a start vertex is chosen to be the vertex with maximal indegree in this component. The present method ensures that, when this start vertex is deleted from the stack, *all* the elementary cycles of this component have been enumerated. Therefore only one start vertex per component is required. As it can be observed from the proposed strategy, if a start vertex would have been chosen to be an arbitrary vertex of the digraph — instead of a vertex with maximal indegree in a strongly connected component — the algorithm could be easily modified so as to avoid finding the strongly connected components. The modified algorithm would have the same time bound as the one currently described.

The basic idea of the algorithm is similar to all previously described methods, namely to try to extend the current elementary path under examination. Consider the case where the content of the stack is $v_1 v_2 \ldots v_{k-1}$ and edge (v_{k-1}, v_k) is reached:

(i) If v_k is not marked then necessarily v_k is not in the stack, the elementary path will be extended with v_k, and an edge from v_k will be examined.

(ii) If v_k is marked and not in the stack then, necessarily, there can be no new elementary cycles generated from the path $v_1, v_2, \ldots, v_{k-1}, v_k$ and therefore v_k is not re-explored, at this stage. Vertex v_{k-1} is inserted in list $B(v_k)$ and v_k is deleted from $A(v_{k-1})$.

(iii) If v_k is marked and lies in the stack then an elementary cycle was found, and it can be recorded at once. The algorithms [21] and [2] also consider this cycle at that stage. However, some efficient algorithms as [19], [3], [7] and [12] disregard it, if v_k is not the start vertex. The problem that arises when considering such a cycle with $v_k \neq v_1$, is that a mechanism for detecting duplicate cycles must be set up. The nature of this mechanism follows from the observation that a cycle is a *new* cycle, if and only if at least one of its vertices had never been deleted from the stack. The fact that it has not been deleted before is indicated by setting a variable q, local to the recursive procedure. For a given computation of this procedure q indicates the top-most vertex of the stack that has never been deleted from it. Therefore, if position $(v_k) \leqq q$ a new elementary cycle is found. Otherwise, this is a duplicate cycle: v_{k-1} is inserted in $B(v_k)$ and v_k is deleted from $A(v_{k-1})$.

In cases (ii) and (iii), when v_k is marked the elementary path is not extended. If a certain elementary path cannot be extended any more, the algorithm backtracks to the previous vertex in the stack, and so on. When the start vertex is deleted from the stack, a new strongly connected component is considered, and so on, until all such components have been processed.

Below is an ALGOL-like formulation of the proposed algorithm. The combined action of variables f and g ensures the correct propagation of the information that a new elementary cycle was found with a certain vertex v at the top of the stack, for all vertices that are below v in the stack.

```
begin comment finding the elementary cycles of a digraph;
  procedure CYCLE (integer value v, q; logical result f);
  begin procedure NOCYCLE (integer value x,y);
    begin insert x in B(y);
      delete y from A(x)
    end NOCYCLE;
    procedure UNMARK (integer value x);
```

```
    begin mark(x) := false;
      for y ∈ B(x) do
      begin insert x in A(y);
        if mark(y) then UNMARK
      end;
      empty B(x)
    end UNMARK;
    logical g;
    mark(v) := true; f := false;
    insert v in the stack;
    t := number of vertices in the stack;
    position(v) := t;
    if ¬ reach(v) then q := t;
    for w ∈ A(v) do
      if ¬ mark(w) then
      begin CYCLE(w, q, g);
        if g then f := true else NOCYCLE(v, w)
      end
      else if position(w) ≦ q then
        begin output cycle w to v from stack then w;
          f := true
        end else NOCYCLE(v, w);
    delete v from stack;
    if f then UNMARK(v);
    reach(v) := true;
    position(v) := N + 1
  end CYCLE;
  read the digraph D;
  A := adjacency lists of the strongly connected components of D;
  for j := 1 step 1 until N do mark(j) := reach(j) := false;
  for each non-trivial strongly connected component do
  begin s := vertex with maximal indegree in this component;
    CYCLE(s, dummy, dummy)
  end
end
```

4. Correctness and Performance.

The following sequence of lemmas and theorems establish the correctness of the method and determine its performance. The proofs have been omitted but they can be established using basically inductive arguments.*

* The proofs can be found in references [22] and [23].

Let $D(V,E)$ be an input digraph with no trivial components:

LEMMA 1. *Every vertex enters the stack at least once.*

LEMMA 2. *If $v_1 \ldots v_k$ constitutes the stack at a given moment and a new elementary cycle is found with v_k then all vertices $v_1,\ldots,v_k$ are unmarked upon leaving the stack.*

LEMMA 3. *Let $v_1,\ldots,v_k,v_1$ be an elementary cycle such that $v_1 \ldots v_k$ or a cyclic permutation of it have already appeared in the k top positions of the stack at some earlier time, and at least one of these vertices has been deleted from it before. If $v_1 \ldots v_k$ now occupy the k top positions of the stack, then all $v_1,\ldots,v_k$ have already been deleted from it.*

LEMMA 4. *Let $z_1,\ldots,z_k$ be an elementary path, (z_k,v) an edge of D, where v is a vertex in the stack that has never been deleted from it. Then if $z_1,\ldots,z_k$ are not in the stack, z_1 is unmarked.*

LEMMA 5. *Let $v_1,\ldots,v_k,v_1$ be a convenient cyclic permutation for an elementary cycle, such that v_1 was the first among v_j, $1 \leqq j \leqq k$ to ever enter the stack. Then there exists a configuration of the stack such that before v_1 leaves the stack for the first time, $v_1 v_2 \ldots v_j$, $1 \leqq j \leqq k$ appear in the j top positions of the stack.*

LEMMA 6. *If a vertex is in the stack, it is marked.*

LEMMA 7. *Each elementary cycle of D is listed at least once.*

LEMMA 8. *Each elementary cycle of D is listed at most once.*

THEOREM 1. *The proposed algorithm for finding the elementary cycles of D is correct.*

LEMMA 9. *If a vertex changes from marked to unmarked twice, a new elementary cycle is enumerated.*

THEOREM 2. *The algorithm requires $O(N+(C+1)M)$ time and $O(N+M)$ space to enumerate C elementary cycles.*

COROLLARY 1. *A time bound per cycle is $O(M)$ for any elementary cycle except for the first enumerated, whose bound is $O(N+M)$.*

5. Critical Remarks.

Prabhaker and Deo [10] have already shown that so far, the most successful cycle-finding algorithms are those based on a backtracking

search strategy. Tiernan's algorithm adopts an essentially unconstrained backtracking. The main difference between the algorithms of Tiernan and Tarjan is that the latter has introduced a marking mechanism which avoids the exploration of a vertex if this vertex is found marked when it is reached. This situation can occur even if this vertex does not lie in the path currently under examination. As a result the backtracking becomes constrained. The basic difference between the algorithms by Tarjan and Johnson is that the latter has modified and improved the marking system. If an elementary cycle is found with a certain vertex v, then upon v leaving the stack, Tarjan unmarks v and all vertices of a set Z which is the set of vertices which are marked, not in the stack, and which entered the stack for the last time, after v. Instead, Johnson unmarks v and only such vertices $z \in Z$ for which there exists a path from z to v, involving solely vertices of Z. Also, all N vertices become start vertices in Tarjan's algorithm. In Johnson's method, for each strongly connected component the number of start vertices equals the number of vertices v such that there exists an edge to v, from a descendant of v in a directed rooted tree, obtained by a depth-first search of this component. These conditions represent further constraints to the backtracking.

The principal difference between Johnson's algorithm and the present one is that we detect an elementary cycle, as soon as it appears in the top positions of the stack. Consequently, while exploring a vertex v we do not seek exclusively cycles involving v and the start vertex, but any other new cycle is considered. Since this earlier detection means that the algorithm will not initiate an explicit new search aimed to find this cycle, as [7] does, this new strategy imposes a further constraint on the backtracking. Also unlike [7] for each non-trivial strongly connected component the present algorithm considers exactly one start vertex. Another difference between the two strategies lies in the marking system: if w is a vertex that is marked and $(v,w) \in E$ then in the proposed method only one unsuccessful exploration of edge (v,w) can occur whilst w remains marked. In [7] each time vertex v is found unmarked, an exploration of edge (v,w) certainly occurs. The effect of these differences in the actual manipulation of digraphs may be appreciated in the following examples. The digraph of Figure 1 has N vertices, $2N-3$ edges and $N-2$ elementary cycles. It has the property that certain vertices (1, 2 and 3 in the example) are involved in every possible existing cycle. Digraphs with this property seem to provide favourable examples for Johnson's algorithm — because if one of these special vertices is the start vertex then each elementary cycle is generated only once. In fact, for such

digraphs both algorithms ([7] and the present) may perform exactly the same number of steps, for identical adjacency lists. In Figure 1 the start vertex is vertex 1 for both algorithms, and both would explore each edge exactly once in the search for the $N-2$ elementary cycles, thus requiring $2N-3$ steps, for termination. Note that by number of steps we mean the frequency of execution of a given statement which has the highest frequency among all by the end of the process (this corresponds to the number of edge explorations). If the digraph is re-labelled such that the new vertex 1 is the previous vertex 2, Johnson's algorithm would take $3N-6$ steps, because the previous vertex 1 (and the edge from it to the new vertex 1) suffers $N-3$ additional explorations. Since this vertex is the vertex with maximal indegree, the present algorithm would always consider it as start vertex and consequently would find all elementary

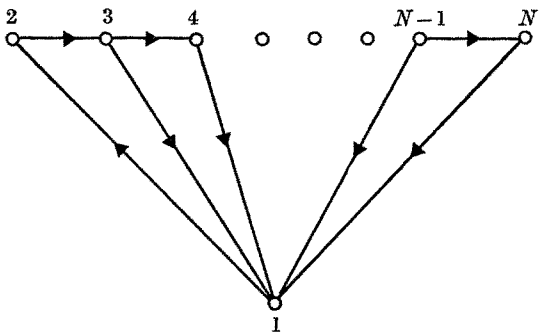

Figure 1.

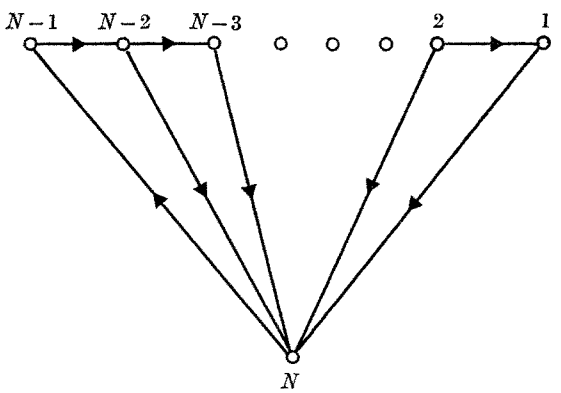

Figure 2.

cycles in $2N-3$ steps. For this class of digraphs, the worst case for Johnson's algorithm occurs when the vertices are labelled as in Figure 2, in which the sub-digraph composed of vertices N, $N-1$, $N-2$, is explored $N-2$ times, the sub-digraph composed of N, $N-1$, $N-2$, $N-3$

is explored $N-3$ times, and so on. A total of $N(N-2)$ steps are required for the enumeration of the elementary cycles of this digraph, using [7] compared with $2N-3$ using the present method.

Concerning the choice of the start vertex we have adopted a different strategy from [7] which always chooses the least vertex as start vertex. Our approach is based on the fact that if $v_1, v_2, \ldots, v_k, v_1$ and $v_1', v_2', \ldots, v_k', v_1'$ are elementary cycles involving precisely the same vertices, $v_1 = v_1'$ and there exists an index j, such that $v_j \neq v_j'$, then this information is sufficient to recognise those cycles as non identical (Johnson has imposed as a further condition — following [20] — that v_1 must be the least vertex of $v_1, v_2, \ldots, v_k$). The alternative that has been adopted in the present method consists of choosing for the start vertex one that is likely not to produce many unfruitful explorations of other vertices in the search for elementary cycles involving the start vertex. If v_1 is the start vertex and v_j is such that $(v_j, v_1) \in E$, then every exploration of v_j leads to a new elementary cycle, hence is not unfruitful. Therefore, the choice for the start vertex to be a vertex with maximal indegree among the vertices of the considered strongly connected component seems to be perhaps more appropriate. Observe that a similar choice could be made as to which vertex to explore, among the vertices $v_2, (v_1, v_2) \in E$ and v_1 the start vertex. Also, this strategy extends to which vertex v_j to explore among the vertices v_j such that $(v_{j-1}, v_j) \in E$, v_{j-1} being the vertex of the top of the stack and not having been deleted from it yet.

Next consider the digraph of Figure 3 with N vertices, $2N-2$ edges and $N-1$ elementary cycles. Johnson's algorithm would consider vertex 1 as start vertex, explore the path $1, \ldots, N$, generate all elementary cycles of the digraph, but since this algorithm only considers cycles involving the start vertex, only the cycle $1, 2, 1$ is enumerated at this stage. Next, vertex 1 is deleted and a similar process occurs for the resulting subdigraph, with vertices $2, \ldots, N$. Vertex 2 is the new start vertex, path $2, \ldots, N$ is again reconsidered, and so on. It takes $N(N-1)$ steps for enumerating all $N-1$ elementary cycles using the above strategy. The present algorithm would find all such cycles in the course of exploring the paths $j, j+1, \ldots, N$ and $j, j-1, \ldots, 1$, where j is the start vertex, consuming precisely $2N-2$ steps, for termination. Digraphs of this class have the additional property that for any start vertex chosen, the present algorithm requires $2N-2$-steps, whilst in [7] there is no possible choice of the start vertex for which the algorithm requires just $O(N)$ steps.

Consider now the complete digraph K_n, with n vertices. Since a new elementary cycle exists with every possible exploration of a given ver-

tex, any vertex is found unmarked, when reached, and this is true for both algorithms. Therefore, in the course of finding the elementary cycles involving the start vertex, all elementary cycles of K_n are generated, but [7] would only enumerate those with the start vertex. Assume now a modified version of [7] with the marking system of the present algorithm incorporated. If T_n is the total number of steps required by the present algorithm to enumerate all elementary cycles of K_n then this modified version of [7] would require $\sum_{j=2}^{n} T_j$ steps for the diagraph K_n.

Figure 3.

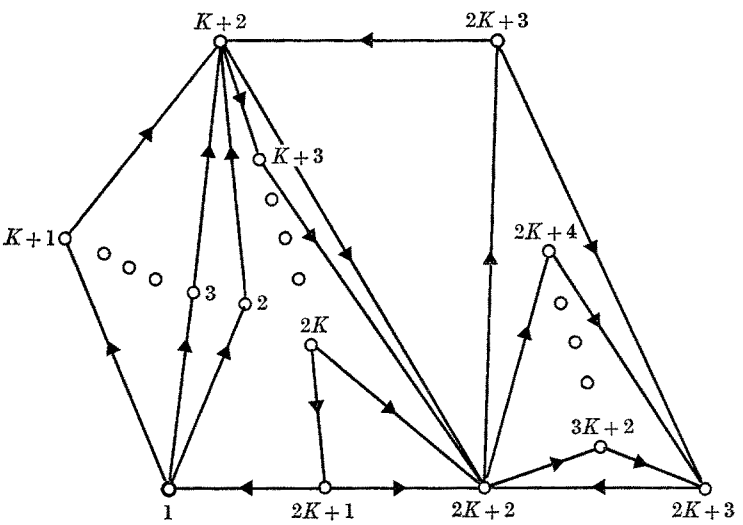

Figure 4.

Consequently, the total number of steps T_n' required by the actual Johnson algorithm for enumerating all elementary cycles of K_n satisfies $T_n' > \sum_{j=2}^{n} T_j$, $n > 2$. Observe, however, that $\sum_{j=2}^{n} T_j$ tends to T_n as n increases.

The example of Figure 4 with $2K+3$ vertices, $6K+2$ edges and $3K$ elementary cycles was shown by Johnson to be a worst case for Tarjan's algorithm, since it realizes the time bound of [19]. The number of steps taken with this input digraph has been decreased by [7] to $O(K^2)$ — more precisely to $6K^2+11K-1$ steps. This value can still be reduced using the present algorithm, which requires $2K^2+6K$ or $7K+1$ steps, depending on which vertex, $K+2$ or $2K+2$ respectively, was chosen

for the start vertex. Note that in this last fortunate case (vertex $2K+2$ the start vertex), each edge of the digraph is explored just once during the entire process, with exception of edge $(3K+3, 2K+2)$ which is explored K times.

6. Conclusions.

An algorithm was presented to enumerate all elementary cycles of a directed graph, based on work by Tiernan–Tarjan–Johnson. Although its (worst case) time bound is similar to that achieved by Johnson, namely $O(N+M)$ per cycle, we believe that the techniques for detecting an elementary cycle, anywhere in the path under examination, and its enumeration at the earliest possible time the cycle is contained in this path, which were used by the present algorithm, represent important features for cycle finding methods.

The present paper has shown examples where unnecessary work was done by existing algorithms. The question that arises is, what about inefficiencies of the proposed algorithm? Clearly, they still exist because a vertex or an edge may be unsuccessfully explored many times, during the process. However, these *same* inefficiencies are also present in the existing backtracking methods. Since we have eliminated some of the inefficiences of those methods, we believe that the proposed algorithm compares favourably with them.

It should be noted that a previous version of the present algorithm [17] was an unsatisfactory attempt to devise a method that would explore unsuccessfully any vertex, at most once during the entire process. An open question still remains about the existence of an algorithm that would find all elementary cycles of a digraph, in such a way that any edge or vertex would be unsuccessfully explored, at most a constant number of times, during the entire process. Such an algorithm would have an optimal time bound.

REFERENCES

1. A. T. Berztiss, *Data Structures: Theory and Practice*, Academic Press, New York, N.Y. 1971.
2. A. T. Berztiss, *A k-tree Algorithm for Simple Cycles of a Directed Graph*, Tech. Rep. 73-6, Department of Computer Science, University of Pittsburgh, Penn., 1973.
3. A. Ehrenfeucht, L. D. Fosdick and L. J. Osterweil, *An Algorithm for Finding the Elementary Circuits of a Directed Graph*, Tech. Rep. #CU-UC-024-73, Department of Computer Science, University of Colorado, Colorado 1973.
4. R. W. Floyd, *Nondeterministic Algorithms*, J. ACM. 14 (1967), 636–644.

5. J. Hopcroft and R. Tarjan, *Efficient Algorithms for Graph Manipulation*, Comm. ACM. 16 (1973), 372–378.
6. J. Hopcroft and R. Tarjan, *Efficient Planarity Testing*, J. ACM. 21 (1974), 549–568.
7. D. B. Johnson, *Finding all the Elementary Circuits of a Directed Graph*, SIAM J. Comp. 4 (1975), 77–84.
8. P. E. Lauer, *The Perils of Indirect Proof or Another Efficient Search Algorithm to Find the Elementary Circuits of Directed Graphs*, Tech. Rep. 42, Computing Laboratory, University of Newcastle upon Tyne, Newcastle upon Tyne, 1973 (revised Sep. 1973).
9. P. G. H. Lehot, *An Optimal Algorithm to Detect a Line Graph and Output its Foot Graph*, J. ACM. 21 (1974), 569–575.
10. M. Prabhaker and N. Deo, *On Algorithms for Enumerating all Circuits of a Graph*, Tech. Rep. UIUCDCS-R-73-585, Department of Computer Science, University of Illinois, Illinois, 1973 (revised Mar. 1974).
11. R. C. Read and R. E. Tarjan, *Bounds on Backtrack Algorithms for Listing Cycles, Paths and Spanning Trees*, Mem. ERL-M433, Electronics Research Laboratory, University of Berkeley, Berkeley, California, 1973.
12. R. C. Read and R. E. Tarjan, *Bounds on Backtrack Algorithms for Listing Cycles, Paths and Spanning Trees*, Networks (to appear).
13. S. M. Roberts and B. Flores, *Systematic Generation of Hamiltonian Circuits*, Comm. ACM. 9 (1966), 690–694.
14. N. D. Roussopoulos, *A* $\max\{m,n\}$ *Algorithm for Determining the Graph H from its Line Graph G*, Inf. Proc. Lett. 2 (1973), 108–112.
15. M. M. Syslo, *Algorithm 459: The Elementary Circuits of a Graph*, Comm. ACM. 16 (1973), 632–633.
16. M. M. Syslo, *Remark on Algorithm 459: The Elementary Circuits of a Graph*, Comm. ACM. 18 (1975), 119.
17. J. L. Szwarcfiter and P. E. Lauer, *Finding the Elementary Cycles of a Directed Graph in* $O(N+M)$ *per Cycle*, Tech. Rep. 60, University of Newcastle upon Tyne, Newcastle upon Tyne, 1974.
18. R. Tarjan, *Depth-First Search and Linear Graph Algorithms*, SIAM J. Comp. 2 (1972), 146–160.
19. R. Tarjan, *Enumeration of the Elementary Circuits of a Directed Graph*, SIAM J. Comp. 3 (1973), 211–216.
20. J. C. Tiernan, *An Efficient Search Algorithm to Find the Elementary Circuits of a Graph*, Comm. ACM. 13 (1970), 722–726.
21. H. Weinblatt, *A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph*, J. ACM. 19 (1972), 43–56.
22. J. L. Szwarcfiter and P. E. Lauer, *A New Backtracking Search Strategy for the Enumeration of the Elementary Cycles of a Directed Graph*, Tech. Report Series, 69, University of Newcastle upon Tyne (1975).
23. J. L. Szwarcfiter, *On Optimal and Near-Optimal Algorithms for Some Computational Graph Problems*, Ph. D. Thesis, University of Newcastle upon Tyne (1975).

UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO,
CAIXA POSTAL 2324 ZC-00,
20.000 RIO DE JANEIRO
RJ – BRASIL

UNIVERSITY OF
NEWCASTLE UPON TYNE
COMPUTING LABORATORY,
NEWCASTLE UPON TYNE,
ENGLAND