An Algorithm for Generating All the Directed Paths and Its Application

Tran DINH Am*, Shuji TSUKIYAMA*, Isao SHIRAKAWA* and Hiroshi OZAKI*

Abstract

An algorithm to generate all the directed paths from a specified vertex to another one in a given graph is proposed. The algorithm requires the processing time bounded by the order $O((n+m)(p+1))$ and memory space bounded by $O(n+m)$, where n, m, and p denote the numbers of vertices, edges, and directed paths in a given graph, respectively. As an application of the algorithm, the shortest or longest path problem in a directed graph containing cycles of negative weight is also considered.

1. Introduction

The problem of finding the shortest path between two specified vertices in a graph has been investigated by a great number of authors (see, for example, [1]). However, any such approach may not be directly applied to an undirected graph with an edge of negative weight or a directed graph (or digraph) with a cycle of negative weight[2]. Generally, to solve this problem for a given digraph containing cycles of negative weight, an algorithm to generate all the directed paths (or dipaths) between two specified vertices might have to be employed in the worst case.

On the other hand, in the computer-aided analysis for a system of graphical structure, the problems of listing all the subsystems satisfying certain particular properties are often confronted. Among these, the problem of listing all the dipaths between two specified vertices is very important in the application of digraph theory, for example, to the reliabilty analysis of a communication network[3].

The approaches so far proposed for generating all the paths may be classified into two: The first is based on the matrix manipulation[4]-[6], the second is on the search techniques[7],[8].

Henceforth, we propose an algorithm with the use of the marking techniques introduced by Johnson[9]. The processing time of the procedure is bounded by $O(n+m)$ per

This paper first appeared in Japanese in Joho−Shori (Journal of the Information Processing Society of Japan), Vol. 16, No. 9 (1975), pp. 774∼780.

* Department of Electronic Engineering, Faculty of Engineering, University of Osaka

dipath, where n and m are the numbers of vertices and edges of a given digraph. Moreover, as an application of the procedure, we also discuss an approach to the shortest dipath problem (or longest dipath problem) for such a digraph as contains cycles of negative weight.

2. Algorithm

Let a digraph G of our interest be a weighted graph, that is, a graph in which a real number w(e) is assigned to each edge e as its weight. Henceforth, for each edge e let s(e) and t(e) represent the _initial_ and _terminal_ vertices of e, respectively, and e denote by an ordered pair $e = (s(e), t(e))$, for which e is said to be _incident from_ s(e) _into_ t(e). Moreover, we assume that graph G does not contain any edge e with $s(e) = t(e)$, and that for any two distinct vertices v and w, G contains at most one edge incident from v into w.

A _dipath_ R of _length_ k from v_0 to v_k is an ordered sequence of edges $[(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)]$ with $v_i \neq v_j$ $(0 \leq i < j \leq k)$, and its _distance_ is the total sum of edge weights of R. Especially when $v_0 = v_k$, R is called a _cycle_ of _length_ k.

The algorithm to find all the dipaths from a specified vertex s to another t in a graph G has been considered by Read and Tarjan[8] with the use of the depth-first search technique. However, in the proposing algorithm, the searches can be made more fruitful by assigning to every vertex the so-called "blocked" or "unblocked" state, as employed by Johnson[9].

The algorithm for generating all the dipaths from a start vertex s to a target vertex t is shown in Fig. 1 in ALGOL-like notations, where we assume that the structure of a given graph is represented by the specification of $E(v) \triangleq \{ e \mid s(e) = v \}$ for each vertex v and a pair of s(e) and t(e) for each edge e.

```
procedure DIPATH GENERATION ;  comment  t is the target and s is the start vertex ;
 begin  list array E(n), B(n) ;  array T(m), w(m) ;  logical array blocked(n) ;
     procedure BACKTRACK ( most recently reached vertex v,  logical result f ) ;
       begin  logical g ;
              procedure UNBLOCK ( blocked vertex u ) ;
                begin
                     blocked(u) := false ;
                     for  y ∈ B(u)  do  begin
                        delete y from B(u) ;
                        if  blocked(y) = true  then  UNBLOCK( y )
                     end
                end  UNBLOCK ;
           f := false ;
           blocked(v) := true ;
           for  e ∈ E(v)  do  begin
```

```
            y := T(e) ;   comment  edge e is incident into y ;
            put e on stack PS ;
            d := d + w(e) ;
            if   y = t   then   begin
                 output of dipath containing in stack PS ;
                 output the distance d of the dipath from s to t ;
                 f := true
            end
            else   if   blocked(y) = false   then   begin
                 BACKTRACK( y, g ) ;
                 if   g = true   then   f := true
            end ;
            delete e from stack PS ;
            d := d - w(e)
         end ;
         if   f = true   then   UNBLOCK( v )
         else   for   e ∈ E(v)   do   begin
                   y := T(e) ;
                   if   v ∉ B(y)   then   put v on B(y)
         end
      end  BACKTRACK ;
   empty stack PS ;
   d := 0 ;
   for   each vertex u of G   do   begin
      blocked(u) := false ;
      B(u) := φ
   end ;
   BACKTRACK( s, flag )
 end  DIPATH GENERATION
```

Fig. 1. An algorithm for generating all the dipaths from s to t.

[Theorem 1] The application of the procedure DIPATH GENERATION yields all the dipaths from s to t without duplication.

[Theorem 2] The procedure DIPATH GENERATION requires memory space bounded by $O(m+n)$ and processing time bounded by $O((m+n)(p+1))$, where n, m, and p denote the numbers of vertices, edges, and dipaths in a given digraph, respectively.

3. Application to the shortest or longest dipath problem

As an application of the algorithm stated above, we consider in the following an approach to the problem of finding the shortest (or longest) dipath.

If graph $\tilde{G}$ is derived from a given graph G by multiplying every edge weight of G by -1, then the problem of finding the longest dipath from s to t in graph G can be reduced to that of finding the shortest dipath from s to t in graph $\tilde{G}$, hence we discuss only the former problem.

Although many efficient procedures have been proposed for this problem[1], they may not be directly applied to a graph permitted the existence of negative cycles. Thus, the proposed algorithm can be considered as an approach to this problem without any such restriction.

However, if we employ such a dipath generation algorithm as a method for the shortest dipath problem, then another aspect of the problem is to be taken into consideration to improve the efficiency.

For two distinct vertices p and q, denote by $\mathcal{R}(p,q)$ a set of all the dipaths from p to q, and by V(R) a set of vertices of dipath R, and then let

$$V(\mathcal{R}(p,q)) \triangleq \bigcup_{R\in\mathcal{R}(p,q)} V(R).$$

If for any vertex x, there holds

$$V(\mathcal{R}(s,x)) \cap V(\mathcal{R}(x,t)) = \{x\},$$

or if for any edge $e=(v,w)$, there holds

$$V(\mathcal{R}(s,v)) \cap V(\mathcal{R}(w,t)) = \phi,$$

then let vertex x or edge e be said to satisfy <u>the splitting condition</u>, respectively.

In applying the proposed algorithm to the shortest path problem, it should be noticed that for any vertex x or edge $e=(v,w)$ satisfying this splitting condition, once an edge incident into x or edge e is explored, the shortest dipath from x or from w to t can be determined. Thus, if any edge incident into x or edge e is explored again, then the edges incident from x or the edges incident from w are no more necessary to be explored. Hence we can see from this observation that the algorithm has some room to be modified so as to be applied to this problem with more efficiency.

Although unfortunately any efficient method to seek all such vertices or edges has not ever been known, a more restricted class of those vertices called <u>dominators</u> or those edges not contained in any strongly connected component, which satisfy the splitting condition, can be sought in processing time bounded by $O(m+n\log n)$ [10] or $O(m+n)$ [11], respectively, and hence we can see that considerable improvements on the algotithm may be attained. For example, suppose that every dipath from s to t passes a vertex x, and let p_1 denote the number of dipaths from s to x and p_2 the number of dipaths from x to t. Then the processing time of $O((m+n)(p_1p_2+1))$ for finding the shortest dipath can be reduced to $O((m+n)(p_1+p_2)+m+n\log n)$.

Furthermore noting that in any acyclic graph (graph without any cycle) every vertex satisfies the splitting condition, the proposed algorithm can be modified into an $O(m+n)$ algorithm for the shortest dipath problem for acyclic graph, which is of great use in the PART problem.

4. Conclusions

This paper proposes an efficient procedure to list up all the dipaths from a specified vertex s to another t in a given graph. Moreover, as an application of this algorithm to the shortest dipath problem, a guideline to reduce the processing time due to some structural consideration on a given graph is observed.

REFERENCES

[1] S.E.Dreyfus, "An appraisal of some shortest path algorithm", Opns. Res., vol. 17, no. 3, pp. 395-412, 1969.

[2] S.Goto and T.Ohtsuki, "The simplex algorithm on a linear graph - A unified view of the extremal path and cutset problems on a graph", IECE of Japan Trans. A, vol. 57, no. 11, pp. 810-817, 1974 (in Japanese).

[3] L.Fratta and U.G.Montanari, "A boolean algebra method for computing the terminal reliability in a communication network", IEEE Trans. Circuit Theory, vol. CT-20, no. 3, pp. 203-211, 1973.

[4] G.H.Danielson, "On finding the simple paths and circuits in a graph", IEEE Trans. Circuit Theory, vol. CT-15, no. 3, pp. 294-295, 1968.

[5] K.Aihara, "An algebraic approach to finding elementary path sets using sparse matrix technique", Mono. IECE of Japan, CST 73-40, Sept. 1973 (in Japanese).

[6] L.Fratta and U.Montanari, "All simple paths in a graph by solving a system of linear equations", IEI. Pisa, Italy, Tech., Note B, Nov. 1971.

[7] P.M.Lin and G.E.Alderson, "Symbolic network functions by a single path finding algorithm", Proc. 7th Allerton Conference on Circuit and System Theory, pp.196-205, 1969.

[8] R.C.Read and R.E.Tarjan, "Bounds on backtrack algorithms for listing cycles, paths, and spanning trees", Networks, vol. 5, no. 3, pp. 237-252, 1975.

[9] D.B.Johnson, "Finding all the elementary circuits of a directed graph", SIAM J. Comput., vol. 4, no. 1, pp. 77-84, 1975.

[10] R.Tarjan, "Finding dominators in directed graphs", SIAM J. Comput., vol. 3, no. 1, pp. 62-69, 1974.

[11] R.Tarjan, "Depth-first search and linear graph algorithms", SIAM J. Comput., vol. 1, no. 2, pp. 146-160, 1972.