論　文

2頂点間のすべての順路の一生成手法とその応用*

トラン・ディン・アム**　築山修治**
白川　功***　尾崎　弘***

Abstract

The present paper proposes a new efficient algorithm to generate all the paths from a specified vertex to another one in a given directed graph. The algorithm requires processing time of order $\theta((n+m)(p+1))$ and memory space of order $\theta(n+m)$; where n, m, and p denote the numbers of vertices, edges, and directed paths to be sought of a graph.

As an application of the algorithm, this paper also considers the shortest or the longest path problem in such a directed graph as contains cycles of negative weight.

1. まえがき

与えられたグラフの，指定された2頂点間の最短経路を求める問題は，これまで極めて多くの著者によって考察された（例えば文献 1),2) を参照）．しかしながら，各辺の重みの総和が負となるようなサイクルを含む有向グラフにおいて同じ頂点を2度以上通らない最短路を見いだす問題や，最長路を見いだす問題については，いままであまり考察がなされていない[3]．一般に，このような何ら制約をうけないグラフにおいてこの種の問題を解く場合には，その性質上指定された2頂点間のすべての順路を見いだすアルゴリズムが必要となるであろう．一方，グラフ的構造を持つシステムの計算機援用解析において，ある指定されたグラフ的性質をもつ部分システムをすべて見い出すという問題に，しばしば直面する．そのなかでも，指定された2頂点間のすべての順路を求める問題は，通信網の信頼度計算[4] など各方面への応用を持ち，グラフ理論の応用という観点からきわめて重要である．

有向グラフにおける指定された2頂点間のすべての順路を見いだす問題に対しては，これまでに多数の著者によって種々の接近法が提案された[5)~12)]．これらは大きく，行列算法を用いた代数的手法[5),6),11)]，木変換を用いた組合せ論的手法[7),10)]，および種々の情報処理的技法をとりいれたいわゆる探索技法[8),9),12)]に類別される．

筆者等はすでに文献 12) により DFS (Depth-First Search) 技法を用いて，2頂点間のすべての順路を $\theta(m\cdot n\cdot p+n+m)$**** の手数で見いだす手法を提案したが，本文では，この手法に基づいて Johnson[13] のマーキング技法を採用しつつ，2頂点間のすべての順路を $\theta((m+n)(p+1))$ の手数で見いだす新しいより能率的な手法を提案する．ただし，ここで n, m, および p はそれぞれ与えられたグラフの頂点，辺，および指定された2頂点間の順路の個数を表わす．本文ではさらに，この手法の応用として，負の重みをもつサイクルが存在するようなグラフにも適用される最短順路問題（または最長順路問題）の一接近法について考察を加える．

2. 諸定義

有向グラフ（以下では単にグラフという）の頂点の集合を $V(G)$，辺の集合を $E(G)$ とし，G を $G=[V(G), E(G)]$ で表わす．本文では，各辺 e に実数の重みが付与されている，いわゆる重みのあるグラフ

---

* An Algorithm for Generating all the Paths between Two Vertices in a Digraph and Its Application by Tran-Dinh-AM, Shuji TSUKIYAMA, (Department of Electronic Engineering, Graduate School of OSAKA University), Isao SHIRAKAWA and Hiroshi OZAKI (Department of Electronic Engineering, Faculty of Engineering, OSAKA University)

** 大阪大学大学院工学研究科電子工学専攻
*** 大阪大学工学部電子工学科

**** $\theta(\cdot)$ は位数関数 (order function) を表わす．

（しばしばネットワークと呼ばれる）を対象とする．以下では，各 $e \in E(G)$ の**始点** $s(e)$，**終点** $t(e)$ に対して，e を $e=(s(e), t(e))$ で表わし，e は $s(e)$ から出て $t(e)$ に入るということにし，さらに一般性を失うことなく，注目するグラフは自己サイクル (v, v) を含まず，かつ任意の $v, w \in V(G)$ に対して v から出て w に入る辺は高々1個であるとする．

グラフ G の辺の順序列 $\sigma=[(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)]$ において，$v_i \neq v_j$ $(0 \leq i < j \leq k)$ であるとき，この σ の各辺から成る G の部分グラフ R を，v_0 から v_k に至る**長さ** k **の順路**といい，かつこれらの各辺の重みの総和を R の**距離**という．一方この順序列 σ において $v_i \neq v_k (1 \leq i < j \leq k)$ かつ $v_0=v_k$ であるとき，この σ の各辺から成る G の部分グラフを**長さ** k **のサイクル**という．

さて，以下では与えられたグラフ G のある指定された頂点 s, t に対して，s から t に至るすべての順路およびそれらのうちで最小の距離をもついわゆる最短順路を求めるアルゴリズムについて考察するが，その際グラフ G の構造は，各 $v \in V(G)$ に対する $E+(v) \triangleq \{e \mid s(e)=v\}$ と各 $e \in E(G)$ に対する $s(e)$ および $t(e)$ の指定によって表現されているものとする．

```
procedure PATH GENERATION; Comment t is the target and s is the start vertex;
  begin list array E+(n), B(n), array S(m), T(m), w(m), logical array blocked(n);
    procedure BACKTRACK (most recently researched 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 path containing in stack PS;
            output the distance of the dipath from s to t;
            f := true;
            end
          else if blocked (y)=false then begin
                 BACKTRACK (y, g);
            L1: if g=false then put v on B(y);
              else f := true;
              end
          else if v∉B(y) then put v on B(y);
          delete e from stack PS;
          d := d-w(e);
          end;
        L2: if f=true then UNBLOCK (v);
      end BACTRACK;
    empty stack PS;
    d := 0
    for u∈V(G) do begin
      blocked (u) := false;
      B(u) := φ;
      end;
    BACKTRACK (s, flag);
  end PATH GENERATION;
```

Fig. 1 Procedure PATH GENERATION.

3. 順路生成アルゴリズム

以下に DFS 技法を用いて与えられたグラフ G の s から t に至るすべての順路と，その距離を求めるアルゴリズムを考察する．DFS 技法を用いた2頂点間のすべての順路を見いだすアルゴリズムとしては，Lin and Alderson[8] などが挙げられるが，これには，不必要な探索（結果的に所望の順路の発見が失敗に終った探索）の占める割合が多くあまり効率的ではない．そこでまず，各頂点に対して blocked および unblocked という2つの状態を割り当て[13]，不必要な探索手続きを可能な限り削減する問題に着目する．

いま，s からある頂点 v に至る一つの順路を R_v とし，v から出る辺 $e=(v, w)$ に注目する．$w (\neq t)$ が R_v 上の頂点ならば，明らかに R_v と辺 e とから成る G の部分グラフは順路とはならない．また，w から R_v の頂点を通らずに t に至るような順路が存在しない場合にも，s から R_v と辺 e を通って t に至るような順路は存在しない．したがって，このような頂点 w を，順路 R_v に対して，blocked の状態として記憶しておけば，この各頂点の blocked の状態の情報によって，不必要な探索の繰り返しを，削減することができる．

つぎに，s から R_v を通って t に至る順路が存在したとし，R_v 上の v に入る辺を $e=(u, v)$ とする．s から u に至る R_v 上の順路を R_u とすれば，R_v に対して blocked である頂点の中には，順路 R_u に対しては blocked であってはならないような頂点も含まれている．それは，その頂点から R_u のどの頂点をも通らずに v に至る順路を持つような頂点である．

このような頂点は，順路 R_u に対して，u から出て行く (u, v) 以外の辺を探索する場合には，blocked の状態を取り除く（すなわち unblocked の状態にする）必要がある．

DFS 技法にこのような機能を付加した順路生成のアルゴリズムを **Fig. 1**（前頁参照）に ALGOL-like な記法で示す．ただし，スタック PS は辺を入れるためのスタックであり，アルゴリズムの各段階で PS が含む辺の系列は，s からの一つの順路を表わしている．この順路を R_{PS} で表わせば，変数 d は順路 R_{PS} の距離 $d(R_{PS})$を示す．また各頂点 $x \in V(G)$ が blocked であるか unblocked であるかをそれぞれ blocked(x)= **true** および **false** で表わし，ある頂点 w が探索されたとき，blocked (w)=**true** であれば，それまでの探索操作によって，s から出てその時点での順路 R_{PS} $(=R_v)$ と辺 (v, w) を通り t に至るような順路は存在しないことが判明した，ということを示している．この論理変数の配列 blocked $(\cdot)$ の値を制御するため，各頂点 v に対して，v に入る辺の始点のリスト $B(v)$ と，順路 R_{PS} 上の各頂点に割り当てられた論理変数 f (flag) を用いる．

また，手続き BACKTRACK (v, f) は，頂点 v から出て行く辺を順次探索するための手続きであり，この手続きの終了時に f が **true** であれば，その時点までに s から出て順路 R_{PS} を通り t に至る順路が少なくとも一つ存在し，それが生成されたことを示している．この場合には，手続き UNBLOCK (v)（ここでは，リスト $B(\cdot)$ を用いた同じ辺を2度以上探索しない DFS 技法が用いられている）によって，blocked であってはならない頂点を unblocked にする操作が，ラベル L2: において行われる．

〔**補題 1**〕 手続き PATH GENERATION のある実行段階において，スタック PS に順路 $R_i=[(s=v_1, v_2), (v_2, v_3)\cdots, (v_{k-2}, v_{k-1})]$ が入っている時点を考える．このとき，与えられたグラフ G に s から出て R_i を通り t に至る順路 $R=[(s=v_1, v_2), (v_2, v_3), \cdots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_k), \cdots, (v_{n-1}, v_n=t)]$ が存在するならば，v_k は blocked ではない，すなわち blocked (v_k)=**false** である．

〈証明〉 k に関する帰納法で証明する．

（I） $k=n$ の場合：$v_k=t$ であり，blocked (t) は **true** となりえないから，補題は明らかに成立する．

（II） $n \geq k' > k$ なるどの k' においても，補題は成立しているものと仮定し，$k'=k$ の場合を考える：アルゴリズムの実行過程の注目する現時点より以前において，v_k に入る辺がスタック PS の先頭に挿入され，手続き BACKTRACK (v_k, f) が呼ばれた時点を考える（このような時点が存在しなければ，blocked (v_k) は **false** のままであるから，補題は明らかに成立する）．このときのスタック PS の順路を R_k とし v_k から $v_n=t$ に至る R 上の順路を R_u とする（**Fig. 2** 参照）．そこで，ある頂点 $x \in V(R_k)-V(R_i)$* に関する手続き BACKTRACK (x, f) のラベル L2: の操作が行なわれる際に，v_k は必ず unblock されることを示そう．このような $V(R_k)-V(R_i)$ に含まれる頂点 x の BACKTRACK (x, f) の L2: の操作は，明らかに BACKTRACK (v_k, f) が呼び出された後で，かつ注目するスタック PS に順路 R_i が入っている時点より以前である．

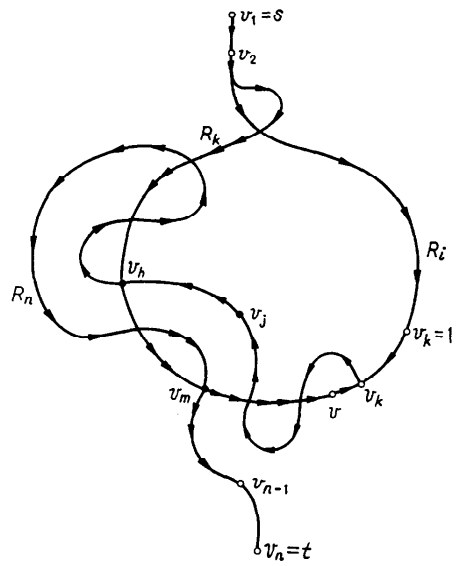

Fig. 2 Illustrating example for the proof of Lemma 1.

（i） R_u と R_k が v_k 以外の頂点を共有しないとき：この場合には，R_k と R_u は s から t に至る1つの順路をなし，帰納法の仮定より，辺 (v_k, v_{k+1}) が探索されたとき，v_{k+1} は blocked ではない．(PS に R_k が入っているとき，辺 (v_k, v_{k+1}) が探索されることは明らかであろう)．それゆえ，辺 $(v_k, v_{k+1}), (v_{k+1}, v_{k+2}), \cdots$, が順次 PS に挿入され，$R_k$ と R_u とから成る順路が必ず見いだされる．したがって，手続き BACKTRACK (v_k, f) の L_2: の命令が実行される

* $V(R)$ は順路 R の頂点集合を示す．

時点では f=**true** となっているから，このとき v_k は必ず unblock され，補題は成立する．

(ii) そうでないとき：共有する頂点のうちで，R_n 上で最も v_n に近い頂点を v_m とし，v_k から R_n 上を v_n に向って進むとき，最初に出会う s から v_m に至る R_k 上の順路の頂点を v_h とする (Fig. 2 参照)．この場合には，s から v_m に至る R_k 上の順路と，v_m から v_n に至る R_n 上の順路は，明らかに s から t に至る順路をなすから，(i) の場合と同様に，BACKTRACK (v_m, f) の L_2: の命令が実行される際には f=**true** になっており，さらに BACKTRACK (v_h, f) の L_2: の命令においても f=**true** となっていることがわかる．BACKTRACK (v_h, f) の L_2: の命令が実行される際 v_k が blocked の状態であるとすれば，このときには，v_k から v_h に至る R_n 上の頂点 $v_j(k \leq j \leq h)$ はすべて blocked であり，$B(v_{j+1}) \ni v_j(k \leq j < h)$ であることが，次の考察よりわかる．辺 (v_j, v_{j+1}) に対して，BACKTRACK (v_j, f) の操作がすべて終った時点で，blocked (v_j)=**true** であるならば，$v_j \in B(v_{j+1})$ である．さらに，BACKTRACK (v_{j+1}, f) の操作がすべて終った時点で，$B(v_{j+1}) \neq \phi$ であるならば，blocked (v_{j+1})=**true** である（これらのことは，アルゴリズムの構造より明らかであろう）．したがって，BACKTRACK (v_h, f) の L_2: の命令が実行される際に，v_k が blocked であれば，手続き UNBLOCK (v_h) によって v_k は必ず unblock され，補題は成立する (証明終)．

〔**定理 1**〕 上記の手続き PATH GENERATION は，与えられたグラフの s から t に至るすべての順路を重複なく見いだす．

〈証明〉 アルゴリズムによって，s から到達しうる頂点から出る辺は，少なくとも一度は探索されることと補題 1 を用いれば，s から t に至るどの順路も必ず見いだされることは明らかであろう．また，スタック PS に辺系列 $\{(s=v_1, v_2), (v_2, v_3), \cdots, (v_{i-1}, v_i)\}$ が現われ，その後この辺 (v_{i-1}, v_i) がいったん取り出されると，決して同じ辺系列がスタック PS に現われることはないから，s から t に至るどの順路もただ 1 度だけ出力される (証明終)．

〔**定理 2**〕 上記の手続き PATH GENERATION は $\theta((m+n)(p+1))$ の演算時間を要し，$\theta(m+n)$ のメモリスペースを要する．

〈証明〉 頂点 v に対するリスト $B(v)$ に入る可能性のある頂点は，v に入る辺の始点のみであることに注意すれば，list array $E^+(u)$ および $B(u)$ に必要なメモリスペースは $\theta(m+n)$ で与えられる．したがって，全体として必要なメモリスペースは $\theta(m+n)$ となる．つぎに，演算時間が $\theta((m+n)(p+1))$ で与えられることを示す．手続き PATH GENERATION が開始されてから s から t に至る最初の順路が見いだされるまでに要する演算時間は，blocked の情報により同じ辺が 2 度以上探索されることはないから，$\theta(m+n)$ である．つぎに，ある順路 $\{(v_1=s, v_2), (v_2, v_3), \cdots, (v_{n-1}, v_n=t)\}$ が見いだされ，この順路上の辺 (v_{i-1}, v_i) がスタック PS から取り出される際に，頂点 x が unblock されたとする．この x が再び blocked された後，再び unblock されるには，その前に必ず新しい s から t に至る順路が見いだされているはずであるから，ある順路が見いだされてから次の順路が見いだされるまでの間にどの頂点も 2 度 unblock されることはない．したがって，この間に，同じ辺が 3 度以上探索されることはないから，ある順路が見いだされてから次の順路が見いだされるまでに要する演算時間は $\theta(m+n)$ で与えられる．さらに，一番最後の順路が見いだされた後，手続き PATH GENERATION が終了するまでに要する演算時間も高々 $\theta(m+n)$ であるから，全体として必要な演算時間は $\theta((m+n)(p+1))$ となる (証明終)．

なお，この手続き PATH GENERATION が無向グラフにも適用可能であることは，容易に理解できるであろう．また，本文では取り扱かってはいないが，一つの頂点 s といくつかの頂点 $t_i(i=1, 2, \cdots, k)$ に

```
for e∈E+(v) do begin
    y := T(e);
    if blocked (y)=false then begin
        put e on stack PS;
        d := d+w(e);
        if y is a target then begin
            output dipath from s to v contained in stack PS;
            f := true;
        end;
        BACKTRACK (y, g);
        if g=false then put v on B(y)
        else f :=t rue;
        d := d−w(e);
        delete e from stack PS;
    end
    else if v∉B(y) then put v on B(y);
end;
if v is a target then f := true;
L2: if f=true then UNBLOCK(v);
```

Fig. 3 The modification for the PATH GENERATION to be applied to the multitarget case.

対して，s から各 t_i な至るすべての順路を求める場合には，この手続き PATH GENERATION を少し変更するだけでよいことを付記しておこう．この場合には，例えば，Fig. 1 の **for** $e \in E^+(v)$ **do begin** の操作以後を **Fig. 3**（前頁参照）に示す手続きで置換すればよいが，これは，blocked および unblocked の性質から容易に類推できるであろう．

4. 最短・最長順路問題への応用

以上考察した順路生成アルゴリズムの応用として，与えられたグラフ G の s から t に至るすべての順路の中で，最小（あるいは最大）の距離を持つ最短順路（あるいは最長順路）を求める問題に対する一接近法について考察する．

与えられたグラフ G の s から t に至る最長順路を見いだす問題は，各辺の重みを -1 倍した重みをもつグラフ $\bar{G}$ の s から t に至る最短順路を見いだす問題に帰着されるから，以下では前者の問題にのみ着目する．

4.1 一般的解法

グラフ G の s から t に至る最短順路問題に対してはすべての辺の重みが非負である場合，あるいは負の重みをもつ辺の存在は許すが重みの総和が負となるようなサイクルが存在してはならない場合に限って多数の効率的なアルゴリズムが報告されている[1),2)]．しかしながら，このような制約を破る一般のグラフに対してはその問題の性質上，最悪の場合には s から t に至るすべての順路を調べる必要があるであろう．この意味において，前述の順路生成アルゴリズムがこの問題の一解法と考えられる．

そこで，手続き PATH GENERATION を最短順路問題の一解法とみた場合に，グラフ G の接続構造がどのようにこの解法の効率化に利用できるか，換言すれば，調べるべき順路の個数を削減するためには，グラフのどのような接続構造に注目すればよいかについて以下で考察する．

相異なる2頂点 $v, w \in V(G)$ に対して，v から w に至るすべての順路の集合を $\mathcal{R}(v,w)$ で表わし，$V(\mathcal{R}(v,w))$ を次のように定義する．

$$V(\mathcal{R}(v,w)) \triangleq \bigcup_{R \in \mathcal{R}(v,w)} V(R)$$

さて，いまグラフ G の頂点 $x(\neq s,t)$ および辺 $e=(v,w)$ に対して，

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

が成立するとき，頂点 x および辺 e はそれぞれ**分割条件**を満足するということにする．

この分割条件を満足する頂点 x（または辺 $e=(v,w)$）に対しては，x を終点とする辺（または辺 e）が一度スタック PS に挿入されれば，それが取り除かれるまでに，x（または w）から t に至る最短順路を決定することができることを意味し，したがって，順路生成アルゴリズムの実行過程でスタック PS に，x を終点とする辺（または辺 e）が再度挿入されたときには，もはや x から出る辺（または e の終点 w から出る辺）について探索を行う必要がなく，手法の効率化をはかる余地がある．

したがって，このような頂点 x および辺 e が，与えられたグラフに多数存在し，かつそれらが容易に見いだされれば，この手法の効率化がきわめて有効に行われる．

この分割条件を満足する頂点または辺を見いだす効率的な手法はまだ知られていないが，しかしこの分割条件の上にさらにある特別の付加的制約が課せられたような頂点や辺に対しては，以下に考察するように比較的簡単に求められる．

いま，s から t に至るすべての順路上に必ず存在するような頂点（いわゆる t の dominator）に注目すれば，これは明らかに分割条件を満たし，Tarjan[14)] の手法により $\theta(m+n\log n)$ の演算時間で求められる．また，どの強連結成分にも含まれていない辺 e も明らかに分割条件を満たすが，これは Tarjan[15)] の手法により演算時間 $\theta(m+n)$ で求めることができる．

したがって，s から t に至る順路が必ずある頂点 x を通ることが判明したような場合には，s から x に至るすべての順路の個数 p_1，x から t に至るすべての順路の個数 p_2 に対して最短順路を見いだすために必要な演算時間を，$\theta((n+m)(p_1 \cdot p_2+1))$ から $\theta((n+m)(p_1+p_2)+m+n\log n)$ へと削減することができる．

そこで，すべての頂点がこの分割条件を満足するアサイクリック・グラフ（サイクルを含まないグラフ）の場合には，どの程度まで効率化できるかを考察する．

4.2 アサイクリック・グラフの場合

アサイクリック・グラフの最短順路問題は PERT 問題などできわめて有用である．これに対しては，上述のように各頂点は分割条件を満たしている．したがって，任意の頂点 v に対して，v を終点とする辺が

一度スタック PS に挿入され，その後それが取り出されるまでに，v から t に至る最短順路を決定することが可能であり，再び v を終点とする辺がスタック PS に挿入された場合，v から出る辺に対して探索を繰り返し行う必要がない．このことに着目すれば，前述の手続き PATH GENERATION に基づいて **Fig. 4** で示すような最短順路を見いだすアルゴリズムを導くことができる．

```
procedure ACYCLIC S-PATH;
    comment procedure for acyclic graph;
    begin logical array searched (u), list array E+(n), array S(m), T(m), w(m);
        procedure DFS (most recently reached vertex v);
            begin
                spd (v) := ∞;
                searched (v) := true;
                for e∈E+(v) do begin
                    y := T(e); comment edge e is incident into y;
                    if searehed (y)=false then DFS (y);
                    if spd (y)+w(e)<spd (v) then begin
                        spd (v) := spd (y)+w(e);
                        link (v) := e;
                    end;
                end;
            end DFS;
    for v∈V(G) do searched (v) := false;
    spd (t) := 0;
    searched (t) := true
    DFS (s);
    v := s;
L1: while v≠t do begin
        e := link (v);
        put e on stack SP;
        v := T(e);
    end;
    comment stack SP denotes shortest dipath from s to t and spd (s) denotes its distance;
end ACYCLIC S-PATH;
```

Fig. 4 Procedure ACYCLIC S-PATH.

ただし Fig. 4 で，論理変数の配列 searched (v) は，頂点 v($\neq t$) を終点とする辺が一度でも探索され手続き DFS (v) が呼びだされたならば* **true**，まだ探索されていなければ **false** の値を取るものとする．また，spd (v) は，アルゴリズムの各段階で見いだされている v から t に至る順路の距離の中での最小の値を示し，link (v) はその最小の値を持つ順路が，v からどの辺を通って出て行くかを示している．また，この手続きの完了時に，スタック SP は，s から t に至る一つの最短順路を示し，spd (s) はその距離を示している．

〔**補題 2**〕 アサイクリック・グラフにおいて，ある頂点 x から t に至る最短順路を P_x とする．もし，y から t に至る最短順路が x を通るならば，この P_x を通る最短順路が存在する．

〈証明〉 x を通る最短順路を P とし，y から x に至る P 上の順路を P_y とする．P_y と P_x とが x 以外の頂点を共有すると仮定すれば，少なくとも1つのサイクルが存在することになり，アサイクリック・グラフの仮定に反する．したがって，P_y と P_x とは，y から t に至る一つの順路をなす．さらに，P_x は x から t に至る最短順路であるから，この P_y と P_x からなる順路は y から t に至る一つの最短順路をなす（証明終）．

〔**定理 3**〕 この手続き ACYCLIC S-PATH の適用により，与えられたアサイクリック・グラフの s から t に至る最短順路を $\theta(m+n)$ の演算時間で求めることができる．

〈証明〉 各頂点 x に与えられた変数 spd (x) の値は，アルゴリズムの実行に伴って，単調に減少し，x から出る辺がすべて探索し終った段階では，補題2からわかるように，x から t に至る一つの最短順路の距離を与える．さらに，この最短順路がどの辺を通るかの情報は，link (x) に登録されているから，s から出る辺がすべて探索され終ったときには，s から t に至る最短順路を決定することが可能となる．これに要する演算時間は，各辺は決して2度以上探索されないことに注目すれば，$\theta(m+n)$ で与えられることは明らかであろう（証明終）．

5. あとがき

本文では，与えられたグラフ G の指定された頂点 s,t に対して s から t に至るすべての順路を $\theta((n+m)(p+1))$ の演算時間で求めることのできる効率的な一手法を提案した．さらにこの手法の応用として最短順路問題へ適用する場合には，グラフの接続状態がその演算時間の削減にどのように利用できるかの指針を述べ，アサイクリック・グラフの場合には，$\theta(n+m)$ の演算時間にまで効率化可能なことを示した．

この順路生成アルゴリズムを，FORTRAN を用いてプログラムし，IBM 370-168 を用いて各種のグラ

* この操作は，v を終点とする辺がスタック PS に挿入され，v から出ていく辺が探索されるということに対応している．

Table 1 Implemented results for complete symmetric graphs

頂点の個数	7	8	9	10	11
順路の個数	326	1,957	13,700	109,601	986,410
CPU time (sec.)	0.026	0.18	1.3	12	115

フについて実験を行ったが，それらのうちこのアルゴリズムに要する演算時間の例として最も典型的な完全対称グラフ* については **Table 1** に示されるような結果を得た．

なお，ここで注意すべきことは，本文では有向グラフの場合について議論を行ったが，この順路生成アルゴリズムは，無向グラフの場合にも容易に適用できるということである．

参 考 文 献

1) S. E. Dreyfus: An appraisal of some shortest path algorithms, Opns. Res., Vol. 17, No. 3, pp. 395～412 (1969).
2) 伊理正夫：ネットワーク問題の理論と手法の最近の進歩，経営科学，Vol. 16, No. 2, pp. 75～87 (1972).
3) 後藤敏，大附辰夫：グラフ理論におけるシンプレックス法—パス及びカットセットの最大最小問題の統一的考察—，信学誌，Vol. 57-A, No. 11, pp. 810～817 (1974).
4) 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).
5) 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).
6) 相原憲一：パスの代数的算出とスパーシティの活用，信学会研資，CST 73-40 (1973. 9).
7) 八星礼剛，篠田庄司：グラフにおけるパスと木の組み合わせ論的考察，信学会研資，CT 71-59 (1972. 1).
8) 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).
9) D. Kroft: All paths through a maze, Proc. IEEE, Vol. 55, No. 1, pp. 88～90 (1967).
10) 有吉弘：リニヤグラフの path 算出について，信学全大，p. 14 (昭43).
11) L. Fratta and U. Montanari: All simple paths in a graph by solving a system of linear equations, IEI Pisa, Italy, Tech., Note B (1971. 11).
12) トラン・ディン・アム，築山修治，白川功，尾崎弘：2頂点間のすべての順路の一生成手法とその応用，信学会研資 CST 74-57 (1974. 10).
13) D. B. Johnson: Finding all the elementary circuits of a directed graph, Tech. Report No. 145, Computer Science Department, Pennsylvania State University, Nov. (1973).
14) R. Tarjan: Finding dominators in directed graphs, SIAM J. Comput., Vol. 3, No. 1, pp. 62～89 (1974).
15) R. Tarjan: Depth-first search and linear graph algorithms, SIAM J. Comput., Vol. 3, No. 1, pp. 146～160 (1972).

(昭和50年3月6日受付)
(昭和50年4月14日再受付)

* 完全対称グラフとはどの相異なる2頂点 u, v に対しても辺 (u, v) (v, u) が一つずつあるようなグラフをいう．