21

OFFPRINTS FROM GRAPH THEORY AND ITS APPLICATIONS TO ALGORITHMS AND COMPUTER SCIENCE
Edited by Y. Alavi et al.

A RECURSIVE ALGORITHM FOR HAMILTONIAN CYCLES IN THE (1,j,n)-CAYLEY GRAPH OF THE ALTERNATING GROUP

Ronald J. Gould*
Emory University

Robert L. Roth
Emory University

ABSTRACT

A recursive algorithm is presented which accepts as input a permutation length $(n \geq 5)$ and a permutation $W \in A_n$ (where W sends $1 \to n$). As output, the algorithm produces a directed Hamiltonian path in the $(1,j,n)$-Cayley Graph D_n of A_n that begins with the vertex representing the identity element of A_n and ends with the vertex representing W. The algorithm makes use of a collection of twelve distinct Hamiltonian paths in D_5.

*Research supported by Emory University Research Grant No. 8399/02

1. Notation.

We use the standard cycle notation for permutations; and the composition, $\Pi_1 \circ \Pi_2$, will always mean Π_1 followed by Π_2. We denote the identity permutation by ι. We often abbreviate $h^{\Pi} = k$ by writing "Π sends $h \to k$". Let A_n denote the alternating group on n letters. We define

$H_n(h_1, h_2,\ldots,h_m; k_1, k_2,\ldots,k_m) =$

$\{\Pi \in A_n : h_r^{\Pi} = k_r,\ r \in \{1, 2,\ldots,m\}\}$. Clearly $H_n(k;k)$ is the stabilizer of k in A_n. The coset $H_n(j;k)$ will often be called the j coset when the value of k is clear from the context. We denote by τ_j the element $(1,j,n)$ and we let $B_n = \{\tau_j : j \in \{2, 3,\ldots,n-1\}\}$. For $n \geq 3$, we let D_n denote the Cayley Graph of A_n with respect to B_n. That is D_n is the directed graph with $V(D_n) = A_n$ and

$E(D_n) = \bigcup_{j=u}^{n-1} E_j$ where $E_j =$

$\{(\Pi,\Psi) : \Pi,\Psi \in A_n$ and $\Pi \circ \tau_j = \Psi\}$. We refer to the elements of E_j as j-edges.

Because $V(D_n) = A_n$, we will feel free to use the terms permutation and vertex interchangeably.

2. Introduction

Recently interest has arisen (see [4], [5], [6], [7] and [2]) in generating a sequencing of the elements of a permutation group subject to various constraints. Of special interest is the problem of generating a sequencing $\Pi_1, \Pi_2,\ldots,\Pi_{|G|}$ of the elements of a permutation group G so that the total cost

$$C = \sum_{i=1}^{|G|-1} c(\Pi_i^{-1} \circ \Pi_{i+1})$$

is minimized, where $c : G \to R^+$ is a cost function. Of course

$\Pi_i \circ \Pi_i^{-1} \circ \Pi_{i+1}) = \Pi_{i+1}$, so that $c(\Pi_i^{-1} \circ \Pi_{i+1})$ is the cost of "proceeding by multiplication" from Π_i to Π_{i+1} in the sequencing.

Tannenbaum, in [6], raised the problem of finding such a sequencing when $G = A_n (n \geq 3)$ with its natural action on $\{1, 2, \ldots, n\}$, $c(\Pi) = |\{j : j^{\Pi} \neq j\}|$ so that $c(\Pi) \geq 3$ for each non-identity element $\Pi \in A_n$, and the set of allowable multipliers for use in the sequencing is $B_n = \{(1,j,n) : j \in \{2,3,\ldots,n-1\}\}$ which is a minimal generating set of A_n. This question was answered in [2]. In this paper we present an algorithm for determining many such sequencings for each $n \geq 5$. Terms not defined in this article can be found in [1] or [3].

3. Overview of the Algorithm

In constructing sequencings for A_n, several results from [2] will be useful. We state them below.

Proposition 1 ([2]). For $k \in \{1, 2, \ldots, n\}$, the left cosets of the stabilizer, $H_n(k;k)$, of k in A_n are precisely the sets $H_n(1;k), H_n(2;k), \ldots, H_n(n;k)$.

When $n \geq 4$ and $k \in \{2, 3, \ldots, n-1\}$, the induced subgraph $\langle H_n(h;k) \rangle$ of D_n is isomorphic to D_{n-1} for each $h \in \{1, 2, \ldots, n\}$. Further, every j-edge of D_n $(j \neq k)$ is an edge of exactly one of these n induced subgraphs, and every k-edge of D_n has its end vertices in two of these induced subgraphs. More precisely, if

$\Pi \in H_n(h_1, h_2, h_3; k, 1, n)$ and $\Pi \circ \tau_k = \Psi$ then,

$\Psi \in H_n(h_2, h_3, h_1; k, 1, n)$.

Proposition 2 ([2]). For $n = 3$ and $n \geq 5$, given any $W \in A_n = V(D_n)$ such that W sends $1 \to n$, there exists a directed $\iota - W$ Hamiltonian path. For $n = 4$, there exists a directed $\iota - (1, 4)(2, 3)$ Hamiltonian path.

We remark that if a directed $\iota - W$ Hamiltonian path exists then a directed $\Pi - \Pi \circ W$ Hamiltonian path exists since premultiplication by any $\Pi \in A_n$ is an automorphism of D_n.

Proposition 3 ([2]). If $n \geq 5$, $\{\Psi \in A_n :$ there exists a directed $\Pi - \Psi$ Hamiltonian path in $D_n\} \supseteq \{\Psi \in A_n : n^{\Psi^{-1}} = 1^{\Pi^{-1}}\}$.

Our goal is to explicitly construct a Hamiltonian path in D_n from the vertex representing the identity element to a specified vertex W, where W sends $1 \to n$ (by Proposition 2 such a path exists).

The algorithm PATHFINDER accepts as input a permutation length $N (N \geq 5)$, the permutation W we wish to be the end vertex of the directed $\iota - W$ path and twelve basepaths in A_5 used in our recursive construction. We require that W be a permutation that sends $1 \to N$. Each of the basepaths is a directed Hamiltonian path in A_5 having initial vertex ι and terminal vertex one of the twelve elements of A_5 which send $1 \to 5$. The output is a directed Hamiltonian path in the Cayley Graph D_N of A_N that begins with the identity element of A_N and ends with W.

The main idea of the algorithm is to use the cosets $H_N(j;k)$ (for a fixed value of k which is moved by W and $j = 1, 2, \ldots, N)$ as much as possible in the construction of the path. Some care must be taken to ensure we do not enter the coset containing W too early in our coset process. It is also important to maintain control of the vertices we traverse first and last in each subgraph representing these cosets. It is fairly easy to convince oneself that it is not possible to traverse all N of the cosets one by one in some fixed order. Thus, we must break out of at least one coset at some time and return later to traverse the remaining vertices in that coset. We choose to begin our path with the identity vertex (which of course is in $H_N(k;k)$) and immediately move across to $(1,k,N)$ (and hence to the coset $H_N(k;N)$). We now traverse all of $H_N(k;N)$ and return to traverse the remainder of $H_N(k;k)$ (see Figure 1).

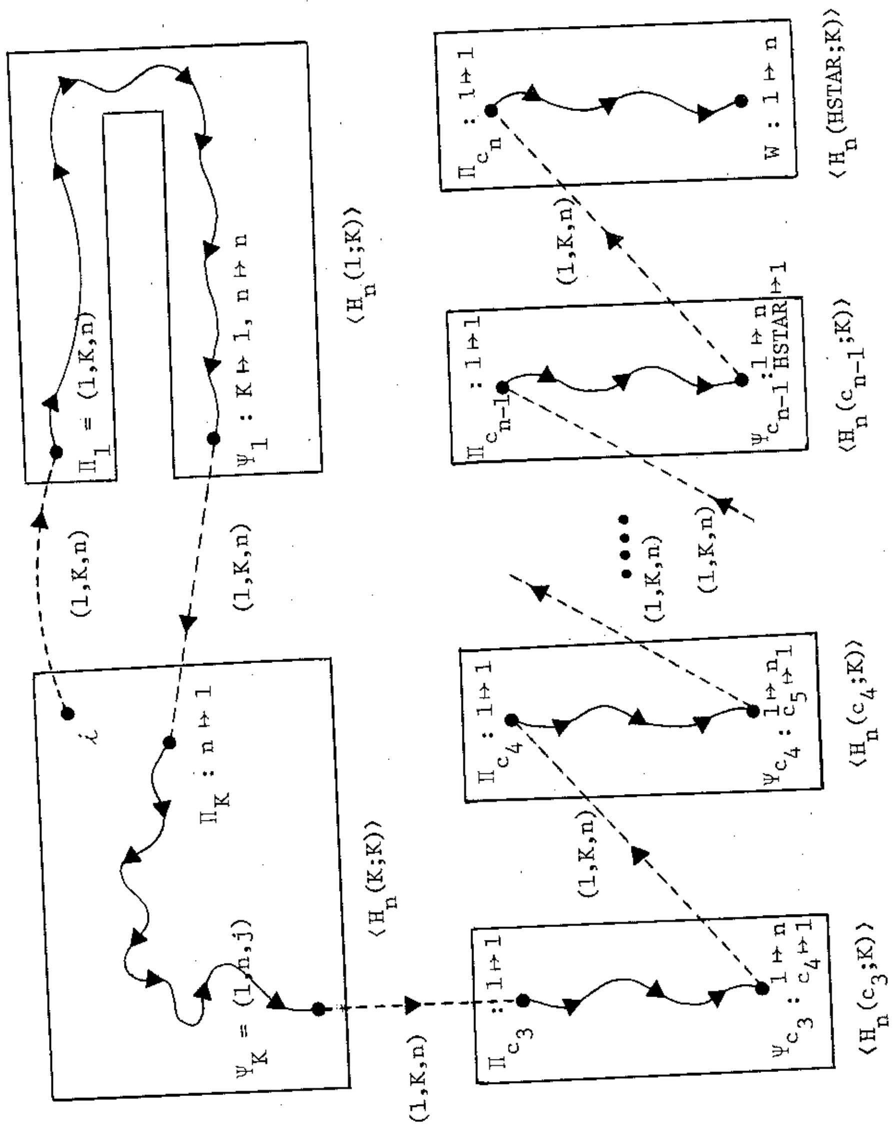
At this point we must determine if a problem exists with the next coset. Since our target vertex W represents a permutation in H_N(HSTAR;k), where HSTAR is the inverse image of K under W, we must be sure at this point that we do not enter this coset next. If on multiplying the terminal vertex of $H_N(k;k)$ by $(1,k,N)$ we determine that our next vertex lies in H_N(HSTAR;k), we modify the path constructed so far to avoid this problem. This is done by conjugating the elements of the path by the involution CYC2 = (HSTAR, GOODCOS3). The values of HSTAR and GOODCOS3 are chosen to avoid entering the HSTAR coset upon leaving $H_N(k;k)$.

Having thus modified our path, we are now able to traverse the remaining cosets, ending with H_N(HSTAR;k), and construct the Hamiltonian path beginning with the identity of A_N and ending with W.

4. The PATHFINDER Algorithm

We now describe the PATHFINDER Algorithm (see Appendix 1) in detail. The PATHFINDER Algorithm relies on the straightforward routines that perform permutation composition (COMPOSE), permutation conjugation (CONJ), upshifting (UPSHFT), downshifting (DWNSHFT), and permutation inversion (INVERSE). Algorithms for these routines are listed in Appendix 2.

If the permutation length N equals 5, then we have recursed to a base case. The path is taken to be the unique one of the twelve basepaths which has W (that is a W suitably



$\Pi_1 = (1,K,n)$
$\Psi_1 : K \mapsto 1,\ n \mapsto n$
$\langle H_n(1;K) \rangle$
$(1,K,n)$
$(1,K,n)$
λ
$\Pi_K : n \mapsto 1$
$\Psi_K = (1,n,j)$
$\langle H_n(K;K) \rangle$
$(1,K,n)$
$\Pi_{c_3} : 1 \mapsto 1$
$\Psi_{c_3} : 1 \mapsto n,\ c_4 \mapsto 1$
$\langle H_n(c_3;K) \rangle$
$(1,K,n)$
$\Pi_{c_4} : 1 \mapsto 1$
$\Psi_{c_4} : 1 \mapsto n,\ c_5 \mapsto 1$
$\langle H_n(c_4;K) \rangle$
$(1,K,n)$
$(1,K,n)$
$\Pi_{c_{n-1}} : 1 \mapsto 1$
$\Psi_{c_{n-1}} : 1 \mapsto n,\ \mathrm{HSTAR} \mapsto 1$
$\langle H_n(c_{n-1};K) \rangle$
$(1,K,n)$
$\Pi_{c_n} : 1 \mapsto 1$
$W : 1 \mapsto n$
$\langle H_n(\mathrm{HSTAR};K) \rangle$

modified by the downshifting described below) as its terminal vertex. The terminal vertex W is compared to the terminal vertices of each of the twelve basepaths in order to find the proper basepath to use for PATH.

If $N > 5$ we begin the coset process. We first compute K, the smallest letter between 2 and $N - 1$ (inclusive) which is moved by W. The value of K is used to determine the coset decomposition of A_N. We next compute HSTAR which is the inverse image of K under W. The HSTAR coset will be the last coset traversed in our coset process since $W \varepsilon H_N(\text{HSTAR};K)$.

We are now ready to begin to construct PATH. First the identity vertex is placed in TMPLST (a temporary storage array for the first two cosets. As noted later, this much of the path sometimes must be modified.) Next the values T1 and T2 are computed. These are the two smallest values in the set $\{2,3,\ldots,N - 1\}$ other than K. The values T1 and T2 are used in computing the first permutation in PSI and the second permutation in PI. The permutations in PI represent the first vertices to be traversed in each of the cosets, while those of PSI represent the last vertices to be traversed in each of the cosets. The first and Kth rows of PI are the permutations (1,K,N) and (1,N)(T1, T2) respectively; while the first and Kth rows of PSI are (1,K)(T1, T2) and the identity respectively.

The permutation WW1 is computed to be the composition of the inverse of the first permutation in PI with the first permutation of PSI. That is, $\text{WW1} = (1,K,N)^{-1} \circ (1,K)(T1, T2) = (1,N)(T1, T2) \varepsilon H_N(K;K)$. The permutation WW2 is the composition of the inverse of the Kth permutation of PI with the Kth permutation of PSI. So $\text{WW2} = (1,N)(T1, T2)$. The permutation WW1 (the terminal vertex of the automorphic copy of the 1 coset) is downshifted to W1, that is, W1 is the permutation of $A_{n - 1}$ acting as $S_K = \{1, 2,\ldots,N\}\backslash\{K\}$ which agrees with

WW1 on S_K. (We observe that K is fixed by WW1.) The routine is then recursively called to obtain a directed Hamiltonian path in D_{n-1} from the identity to W1. This path is returned as SMLPATH. The vertices of SMLPATH are now successively upshifted to A_N (being stored in UU), premultiplied by the first row of PI (to obtain a path beginning at (1,K,N) and ending at (1, K)(T1, T2)) and placed in TMPLST. We now repeat the actions taken on WW1 for WW2 and the K coset except that we omit the last vertex of the path obtained (since it is the identity, which has already appeared).

We have now traversed the first two cosets of A_N in our process (see Figure 1). At this point some care must be taken. We must check the last vertex of TMPLST and if N is sent to HSTAR under this permutation, then the path must be modified to free the HSTAR coset to be the last one traversed. (We recall that $W \varepsilon H_N$ (HSTAR;K).) Thus, in this case we conjugate all of the vertices in TMPLST using the involution (HSTAR, GOODCOS3). The value of GOODCOS3 is set to be the smallest value in {2, 3,...,N - 1} other than K and HSTAR. The involution CYC2 is set equal to (HSTAR, GOODCOS3) to perform the conjugation. The vertices of TMPLST are then conjugated (if necessary) and stored in PATH. In any case, GOODCOS3 is computed to be the image of N under the permutation which represents the last vertex presently in PATH.

The distinct values K, GOODCOS3 and HSTAR are sorted in ascending order and stored in BAD1, BAD2 and BAD3. These represent the distinguished cosets that must be treated first, third and last in the coset process. We now place these and the remaining N - 3 cosets into COSET, which is a listing of the order in which the cosets will be traversed.

The COSET(3)-th row of PI is computed to be the permutation

(K,N,COSET(3)) = (1,N,COSET(3)) ∘ (1,K,N) which is correct since (1,N,COSET(3)) is the last permutation presently in PATH (whether or not conjugation by CYC2 was necessary) because it (or its pre-image under the conjugation by CYC2) was the penultimate vertex in the traversal of the K coset which terminated with the identity.

Next, for I = 3, 4,...,N - 1, the COSET(I)-th row of PSI and the COSET(I+1)-th row of PI are computed. The COSET(I)-th row of PSI is computed to be (1,N)(COSET(I),K) if N = COSET(I+1), (1,N,K,T4,COSET(I+1)) (where T4 is the smallest letter different from I,N,K, and COSET (I+1)) if N = COSET(I) and (1,N,COSET(I),K,COSET(I+1)) otherwise. The COSET(I+1)-th row of PI is computed to be (K,COSET(I),N) if N = COSET(I+1), (K,T4,COSET(I+1)) if N = COSET(I), and (K,COSET(I+1))(COSET(1),N) otherwise.

Finally, the COSET(N)-th row of PSI is computed to be W.

We are now ready to generate the paths in the remaining cosets. Their order, beginning vertices and ending vertices have been computed to allow us to move from a completed coset to the next coset via multiplication in A_N by (1,K,N). Writing c_i for COSET(I) we see that for each $H_N(c_i; K)$, $i = 3, 4,\ldots,N$, the directed Hamiltonian path constructed has initial vertex Π_{c_i} (the c_ith entry of PI) which sends $1 \to 1$ and for $i = 3, 4,\ldots,N - 1$ this path has terminal vertex Ψ_{c_i} (the c_ith entry of PSI) which sends $1 \to N$ and $c_i \to 1$. Thus $\Psi_{c_i} \circ (1,K,N) = \Pi_{c_i + 1}$ as is required.

For the remaining cosets, WWL is computed to be the composition of the inverse of the c_Lth entry of PI with the c_Lth entry of PSI. That is, WWL is the terminal vertex for the automorphic image of the c_Lth coset. As before, WWL is downshifted to WL in A_{N-1} and the PATHFINDER routine is recursively called to obtain a directed Hamiltonian path from the identity to WL in A_{N-1}. This path is returned as SMLPATH and the vertices of SMLPATH are successively upshifted to A_N (and stored in UU), premultiplied by the c_Lth entry of PI and placed into PATH.

APPENDIX 1

```
/*  This is the PATHFINDER Algorithm described in section 4.
Input to the algorithm is the permutation length  N,  the final
permutation  W  and the basepaths in the group  A(5).  Output from
the algorithm is a Hamiltonian path in PATH in the group  A(N)
that begins with the identity and ends with  W.  The Algorithm
is written in a C-like pseudocode. */

PATHFINDER (N, W, PATH)
/* Compare the downshifted  W  to the end vertices of the base-
paths to choose the correct path.  */
if  (N=5) {   I = 1;
          while (I<= 12) {  IT = I;
              if(W does not equal BP[60][I]) { I += 1; }
              if ( I=IT )  {  for (II = 1; II<=60; ++II) {
              PATH[ II ] = BP[II][I]; }
              I=13
          }
 }
 return;
 }
 /* We begin the coset process by finding  K  and  HSTAR */
 else  {  I=2;
          while (2<=I and I<=N-1) {
            if (W does not send I -> I) { K=I; I=N; }
            else I = I + 1;
          }

          I=2;
          while( 2<=I and I<= N ){
            if (W sends I -> K)   { HSTAR = I; I = N+1; }
            else I = I + 1;
    }
```

```
/* Place the identity first, then find  T1  and  T2; the smallest
values other than  K. */
  TMPLST[1] = identity;
        if ( K = 2 )   { T1=3; T2=4; }
        else {   T1=2;
                 if (K=3) T2=4;
                 else T2=3;
        }
 /*  Store PI and PSI for the 1 and K cosets. */
   PI[1]=(1,K,N); PI[K]=(1,N)(T1,T2);
   PSI[1]=(1,K)(T1,T2); PSI[K]=identity;

 /*  Compute WW1= PI(1)^-1 o PSI(1)   */
   inverse (N,PI[1],PI1INV) ;
   compose (N,PI1INV, PSI[1], WW1) ;

 /*  Compute WW2  in a similar manner. */
   inverse (N,PI[K],PIKINV) ;
   compose (N, PIKINV, PSI[K], WW2) ;

 /*  Downshift to A(N-1) and recurse. */
   dwnshft (N, K, WW1, W1) ;
   PATHFINDER (N-1, W1, SMLPATH) ;

 /*  Rebuild the vertices by upshifting to A(N), premultiplying by
 PI(1) to adjust the path, and store the vertices. */
   for (I=1; I<= FACT[N-1]/2; ++I) {
          upshft (N-1, K, SMLPATH[I], VV);
          compose (N, PI[1], VV, PI1VV) ;
          TMPLST[I+1] = PI1VV ;
   }
```

```
/*  Repeat the process on WW2. */
  dwnshft (N, K, WW2, W2) ;
  PATHFINDER (N-1, W2, SMLPATH) ;
  for (I=1; I<=FACT[N-1]/2 -1; ++I) {
        upshft (N-1, K, SMLPATH[I], VV) ;
        compose (N, PSI[K], VV, PIKVV) ;
        TMPLST[  FACT[N-1]/2 + I+1] = PIKVV;
   }
/* Determine if the third coset must be adjusted to avoid the HSTAR
coset.  Then set  CYC2 for conjugating and adjust the temporary
path.
*/
   if (TMPLST[ FACT[N-1]] sends N -> HSTAR )  {  I=2;
        while (2<=I and I<= N {
                 if (I does not equal K and I does not equal HSTAR)
                    {GOODCOS3 = I; I = N+1;}
                    else I = I + 1;
    CYC2 = (GOODCOS3,HSTAR);
      for ( I=1; I<= FACT[N-1]; ++I) {
          conj (N, CYC2, TMPLST[I], PATH[I] ;
      }
    }
  /*  No adjusting, just copy to PATH, then get GOODCOS3.  */
    else  { for (I=1; I<= FACT[N-1]; ++I) {  PATH[I] = TMPLST[I]; }
    GOODCOS3 = the image of N  in TPMLST[FACT[N-1]];
    }
  /*  Build the coset order for processing.  */
    BAD1 = K;  BAD2 = GOODCOS3;  BAD3 = HSTAR;
    SORT(BAD1,BAD2,BAD3);
    COSET[1] = 1;  COSET[2] = K;  COSET[3] = GOODCOS3;
    COSET[N] = HSTAR;
  J = 2;   JJ = 3;
  while (2<=J and J<= N) {
        if (J = BAD1) J += 1;
        if (J = BAD2) J += 1;
        if (J = BAD3) J += 1;
        JJ += 1;
        if(J<=N) COSET[JJ] = J;
        J += 1;
   }
/*  Construct the PI and PSI permutations for the remaining
    cosets. */
   PI[COSET[3]] = (K,N,COSET[3]);

   for (I=3; I<= N-1; ++I) {
     if(COSET[I+1]=N {  PSI[COSET[I]] = (1,N)(COSET[I],K);
                        PI[COSET[I+1]] = (K,COSET[I],N);
     }
   else {
     if ( COSET[I] = N {
         for(J=2; J<=N-1; ++J) {
            if(J does not equal K and J does not equal COSET[I+1])
                 { T4 = J; J =N; } }
         PSI[COSET[I]] = (1,N,K,T4,COSET[I+1]);
         PI[COSET[I+1]] = (K,T4,COSET[I+1])
     }
     else { PSI[COSET[I]] = (1,N,COSET[I],K,COSET[I+1]);
            PI[COSET[I+1]] = (K,COSET[I+1])(COSET[I],N); }
    }
    }
    PSI[COSET [N]] = W;
/*  Now construct paths in the remaining cosets in a manner
similar to that used on the special cosets.  COSET determines the
order.  */
        for (L=3; L<=N; ++L){
                inverse (N, PI[COSET[L]], PICLINV);
                compose (N, PICLINV, PSI[COSET[L]], WWL) ;
                dwnshft (N, K, WWL, WL);
                PATHFINDER (N-1, WL, SMLPATH);

                for (I=1; I<= FACT[N-1]/2; ++I) {
                upshft (N-1, K, SMLPATH[I], VV);
                compose (N, PI[COSET[L]], VV, PICLVV);
                PATH[((L-1)*FACT[N-1]/2 +I)] = PICLVV:
                }
         }
  return;
  }
```

APPENDIX 2

```
/*  ROUTINE INVERSE to compute the inverse of a permutation of
length N.  The routine accepts a permutation length N, a permu-
tation alfa, and returns alfa's inverse in invalfa.  Both alfa
and invalfa are thought of as arrays of length N.  */

inverse ( N , alfa , invalfa )
{
for (I=1; I =N; ++I)   invalfa[ alfa[I] ] = I;
return;
}

/*  ROUTINE COMPOSE to multiply permutations of length N.  The
routine accepts a permutation length N and permutations alfa and
beta (thought of as arrays of length N).  The composition is
returned in athenb.  That is, athenb = alfa o beta.  */

compose ( N, alfa, beta, athenb)
{
for (i=1; i =N; ++i)   athenb[i] = beta[ alfa[i] ];
}

/*  ROUTINE CONJ to conjugate permutations.  This routine accept
a permutation length N and two permutations of length N, namely
alfa and beta.  The routine computes the conjugation of beta by
alfa.  That is, iaba = invalfa o beta o alfa.  Note the temporar
variables iab and iaba to pass returned values.  */

conj (N,alfa,beta,iaba)
{
inverse(N,alfa,invalfa);
compose(N,invalfa,beta,iab);
compose(N,iab,alfa,iaba);
}
```

```
/*  ROUTINE DWNSHFT to downshift a permutation to one in A(n-1).
The routine accepts a permutation length N, a specified value k,
and a permutation of length N in VV.  The permutation to be
downshifted is VV and the returned permutation of length N-1 is
V.  */

dwnshft(N,k,VV,V)
{
for( i=1; i<= k-1; ++i)  {
         if (VV[i] < k)      V[i] = VV[i];
         else   V[i] = VV[i] - 1;
}
for(i=k+1; i<=N; ++i)    {
         if (VV[i] < k)      V[i-1] = VV[i];
         else    V[i-1] = VV[i]-1;
}
return;
}

/*  ROUTINE UPSHFT to reverse the process of dwnshft and insert
the k -> k element of a permutation of length N-1 to build it to
a permutation of length N.  The routine accepts a permutation
length N, a value k, and a permutation to rebuild V (thought of
as an array of length N-1).  The routine returns the rebuilt
permutation in VV, a permutation of length N.  */

upshft(N,k,V,VV)
{
for (i=1; i<= k-1; ++i)   {
        if (V[i] < k)      VV[i] = V[i];
        else    VV[i] = V[i]+1;
}
VV[k] = k;
for(i=k+1; i<= N+1; ++i)   {
        if(V[i-1] < k)     VV[i] = V[i-1];
        else    VV[i] = V[i-1] +1;
}
return;
}
/*  ROUTINE SORT, a routine to sort three values into ascending
order.  Any sorting technique will work here.  */
```

REFERENCES

1. M. Behzad, G. Chartrand, and L. Lesniak-Foster, "Graphs and Digraphs", Wadsworth International Mathematics Series, 1979.

2. R.J. Gould and R. Roth, Cayley Graphs and (1,j,n)-Sequencings of the Alternating Group, preprint.

3. J.J. Rotman, "The Theory of Groups (Second Edition)", Allyn and Bacon Series in Advanced Mathematics, 1973.

4. R. Sedgewick, Permutation Generation Methods, Computer Surveys 9(1977), 137-164.

5. P.J. Slater, Generating all Permutations by Graphical Transportations, Ars Combinatoria 5(1978), 219-225.

6. P. Tannenbaum, Minimal Cost Permutation Generating Algorithms, Proceedings of the Fourteenth Southeastern Conference on Combinatorics, Graph Theory, and Computing, to appear.

7. D.S. Witte, On Hamiltonian Circuits in Cayley Diagrams, Discrete Mathematics 38 (1982), 99-108.