Paper 66-30

The Pegboard Game: An Iterative SAS® Program to Solve It

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

The well-known pegboard game has been a fixture at restaurants to entertain customers, and I've never found that I could reduce the number of pegs to one. I decided to produce a SAS program to compute the necessary permutations and even added some extra twists. This presentation describes the algorithm.

PEGBOARD GAME

Every time I visit one of the Cracker Barrel restaurants, I encounter the Pegboard Game. (See the SAS/GRAPH image of the Pegboard Game in Figure 1. The SAS code at the end of this paper shows how the image was created.) This game consists of a triangular board with 15 holes and 14 pegs (which look like golf tees) in the holes. The object of the game is to jump over pegs (just like you jump over marbles in Chinese Checkers) and end with only one peg.

I've never been able to end with just one peg—occasionally three, but never two, and definitely never one. I was beginning to wonder if the problem was unsolvable.

So I decided to try a little exercise using the DATA step in SAS to see if there's a solution. The exercise was also quite useful to show how to approach recursion problems in a non-recursive language such as the DATA step.

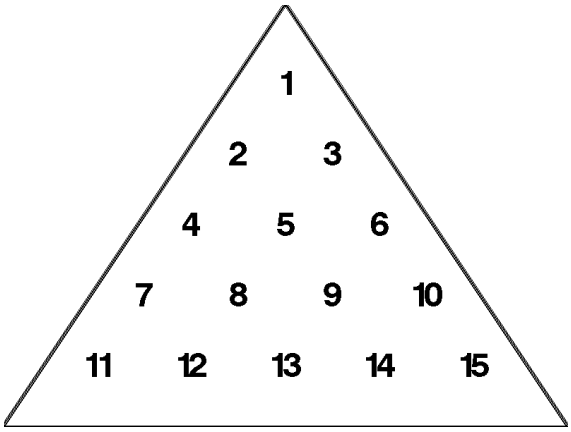

Figure 1. A SAS/GRAPH Image of the Pegboard Game

I started by numbering the holes (as shown in Figure 1). Next, I produced a list of all the possible jumps. The jump definitions consist of the starting hole, the hole that's jumped over, and the hole that's jumped to. Notice that the jumps can go in both directions, so I supplied the jump configurations one way, and programmed a swap of hole 1 and hole 3 so that both combinations would be output. Here is the SAS code to compute the possible jumps.

```
data jumps;
     input j1-j3 @@;
     output;
     k=j3; j3=j1; j1=k; output;
     drop k;
     datalines;
 1  2  4   2  4  7   4  7 11   1  3  6   3  6 10   6 10 15   4  5  6
 7  8  9   8  9 10  11 12 13  12 13 14  13 14 15   2  5  9   3  5  8
 5  9 14   5  8 12   4  8 13   6  9 13
;

run;
```

You need a macro variable to indicate the number of observations in the JUMPS data set. I used the following short DATA step to create the macro variable *njumps*.

```
data _null_;
     call symput('njumps',trim(left(put(nobs,best12.))));
     stop;
     set jumps nobs=nobs;

run;
```

The following is the primary DATA step for computing the possible solutions. The approach is to use a "stack" of board layouts. A board layout is a 15-byte character string in which each character is either a 1 or a 0. If a character is a 1, this means that the corresponding hole is occupied; if a character is a 0, the hole is not occupied. For example, if a board layout is '001001001010010', this means that holes 3, 6, 9, 11, and 14 have pegs in them, and the remaining holes are empty.

Each move results in a new board layout. Because there are 15 holes, there will be 14 different board layouts until there is only one hole filled. With each move, we "stack" the layout by adding the layout to the board array and incrementing the index. You keep adding to the stack until you find that the board layout does not allow any more moves. If there are no more possible moves and the board layout has only one peg left, then you found a solution.

To create a board layout, copy the previous board layout and look at all the possible moves on the board. Run through all the jumps (via the POINT= option), and see if a jump can be executed on the board. If it can, then create a new board layout based on the result of that jump, stack the layout, and continue. Also, keep a record of the jump number for each board layout, so that you can move through all possible jumps at each of the 14 board layouts.

The DATA step begins by indicating which variables should be saved. MOVES is a character string that lists all the jumps. HOLE is the number for the starting hole. The POSS array has a row for each board layout and a column for each possible jump for that board layout. The WHICH array has an element for each board layout that indicates which jump is currently being processed. The BOARD array has an element for each board layout in the stack.

```
data jumps(keep=moves hole);
     array poss(14,&njumps) _temporary_;
     array which(14) _temporary_;
     length board1-board14 $15;
     array board(boardnum) board1-board14;
     do i=1 to 14; which(i)=0; end;
```

Now, loop through all possibilities for each starting hole. Even though there are 15 holes on the board, there are actually only four different unique starting-hole positions. For example, hole 3 is the mirror image of hole 2. All moves that start with hole 2 are mirrored by the moves of hole 3, so they're not really "different" moves. Likewise, if you rotate the board clockwise so that hole 11 is now the topmost hole, then hole 12 occupies the same spot that hole 2 once occupied. This means that any moves that start from hole 12 mimic the moves that start from hole 2.

With this in mind, only holes 1, 2, 4, and 5 need to be processed as starting holes.

For each hole, start the board stack with all 1s except for the single hole.

```
       do hole=1,2,4,5;
          boardnum=1;
          ntries=0; nsuccess=0;
          board='111111111111111';
          substr(board,hole,1)='0';
          boardchg=1; firstone=0;
```

Continue to loop until the board stack is back to the bottom. This means that all possible combinations were tried with the starting hole.

```
       do while(boardnum>0);
```

The BOARDCHG variable indicates that a move was made in the previous iteration. This means that you need to determine all the possible jumps with the new board layout. So, you run through all the observations in the JUMPS data set to see if each jump is possible with the current board layout. Record the OBSNUMS in POSS and the total number of possible jumps in WHICH.

```
          if boardchg then do;
             jj = 0;
             do i=1 to &njumps;
                link getobs;
                if substr(board,j1,1)='1' and
```

```
            substr(board,j2,1)='1' and
            substr(board,j3,1)='0' then do;
            jj+1;
            poss(boardnum,jj)=i;
            end;
         which(boardnum)=jj;
         end;
      end;
```

If you determine that there are no possible jumps, count how many pegs are left. The COMPRESS function is used to remove the 0s, and the resulting LENGTH of that string tells us how many 1s are present. If there is exactly 1 (meaning one peg is left), then you found a combination of moves that ends in one peg, and that combination will be output. Then, regardless of peg count, you "pop" the stack because all the moves for this board layout were tried. Now, go back to the previous layout to try more moves.

```
if boardchg and jj=0 then do;
   nleft=length(compress(board,'0'));
   if nleft=1 then do;
      link putout;
      end;
   else ntries+1;
   link pop;
   end;
```

If you've run through all the possible moves for the current layout, then “pop” the stack again.

```
   else if which(boardnum)=0 then link pop;
```

If you get this far then you have found a board layout that has moves left. You know that POSS contains the jump number, and WHICH contains the remaining jump count. So, you get the jump number, decrement the possible jump count, "push" the current board onto the stack, and apply the jump to the board. Now, you're ready to continue looping with the new board.

```
else do;
   i = poss(boardnum,which(boardnum));
   which(boardnum)=which(boardnum)-1;
   tempbrd=board;
   boardnum+1;
   board=tempbrd;
   link getobs;
   substr(board,j1,1)='0';
   substr(board,j2,1)='0';
   substr(board,j3,1)='1';
   boardchg=1;
   end;
end;
```

At the bottom of the loop, you can report how many unsuccessful tries were made along with the number of successful tries.

```
put hole= ntries= nsuccess=;
end;
```

The STOP statement will be needed to stop processing due to the use of the POINT= option. The RETURN statement will be needed to avoid going into the LINK routine code.

```
stop;
return;
```

The following LINK routine will pop the board stack. To do this, decrement the board number and see if there are still

jumps to be checked for the previous board.

```
pop:;
     do i=boardnum-1 to 1 by -1 while(which(i)=0); end;
     boardnum=i;
     boardchg=0;
     return;
```

The following LINK routine simply reads the jump value that's requested.

```
getobs:;
     set jumps point=i;
     return;
```

The following LINK routine outputs the observations. The observations have a string variable named MOVES, which contains the list of moves. The list of moves is in the POSS array. Specify the j1-j3 values, producing a character string of the form 1-2-3, 4-5-6, etc..

```
putout:;
     length moves $150;
     moves=' ';
     do jj=1 to boardnum;
        i = poss(jj,which(jj)+1);
        link getobs;
        moves=trim(moves)||
              compress(put(j1,2.)||'-'||put(j2,2.)||'-'||put(j3,2.))||
              ',';
        end;
     substr(moves,length(moves),1)=' ';
     output;
     nsuccess+1;
     return;
run;
```

RESULTS OF THE PROGRAM

So what does this program tell us?

- For starting hole 1, there were 29760 successful combinations and 538870 unsuccessful attempts.
- For starting hole 2, there were 14880 successful combinations and 279663 unsuccessful attempts.
- For starting hole 4, there were 85258 successful combinations and 1064310 unsuccessful attempts.
- For starting hole 5, there were 1550 successful combinations and 136296 unsuccessful attempts.

There are a total of 131448 different combinations of moves that will result in having one peg remain, and 2019139 different combinations were unsuccessful. Only 6% of the possible combinations are successful in having a single peg remain. This explains why random attempts rarely result in a single-peg completion.

An interesting side note is the idea of ending with a single peg in the starting hole, which is possible for holes 1, 2, and 4 but not for hole 5. Here are combinations for holes 1, 2, and 4 that end with a single peg in the starting hole.

```
(hole 1): 6-3-1,13-9-6,2-5-9,15-14-13,12-13-14,10-6-3,1-3-6,6-9-13,
          14-13-12,11-12-13,7-4-2,13-8-4,4-2-1,4-2-1
(hole 2): 7-4-2,13-8-4,3-5-8,15-14-13,12-13-14,10-6-3,1-3-6,6-9-13,
          14-13-12,11-12-13,2-4-7,13-8-4,7-4-2,7-4-2
(hole 4): 13-8-4,6-9-13,2-5-9,14-9-5,12-13-14,15-14-13,1-3-6,10-6-3,
          3-5-8,7-4-2,13-8-4,2-4-7,11-7-4,11-7-4
```

Even though no starting-hole 5 combination will end in hole 5, here's one of the starting-hole 5 combinations:

```
(hole 5): 12-8-5,14-13-12,6-9-13,2-5-9,12-13-14,15-10-6,3-6-10,10-9-8,
          7-4-2,1-2-4,4-8-13,14-13-12,11-12-13,11-12-13
```

SAS CODE FOR PRODUCING PEGBOARD IMAGE

```
   /*-----------------------------------------------------------------  */
   /* Produce the coordinates for the 15 holes. The holes are numbered    */
   /* from 1 to 15, starting with 1 at the top of the triangle and        */
   /* increasing by 1 left-to-right down the triangle. You want a label  */
   /* for each hole that is the hole number. Use row numbers 5 to 1 for Y  */
   /* and column numbers 1 to 5 for X. The FUNCTION value will be LABEL   */
   /* and the TEXT value will be the hole number. This is in preparation  */
   /* for processing by the GANNO procedure.                              */
   /*-----------------------------------------------------------------  */

  data temp;
       function='LABEL';
       length text $2;
       pegnum=1;
       startx=3; nx=1;
       do y=5 to 1 by -1;
          x=startx;
          do i=1 to nx;
             text=trim(left(put(pegnum,2.)));
             output; x+1; pegnum+1;
             end;
          startx=startx-.5;
          nx+1;
          end;
       keep function text x y;
   run;

   /*-----------------------------------------------------------------*/
   /* Create a border for the triangle. The border will be one unit off */
   /* for both X and Y. The function is DRAW because lines are drawn for  */
   /* the border.                                                       */
   /*-----------------------------------------------------------------*/

  %let minx=1; %let maxx=5; %let miny=1; %let maxy=5;
  data border; set temp2;
       function='DRAW ';
       y=&maxy+1; x=((&maxx-&minx)/2)+&minx; output;
       y=&miny-1; x=&maxx+1; output;
       x=&minx-1; output;
       keep function text font x y;
  run;

   /*-----get the max values for percentage calculation----------------*/
  proc means data=border(keep=x y) noprint;
       var x y;
       output out=xxx max=maxx maxy;
  run;

   /*-----set up to write out the GIF file-----------------------------*/
  filename grafout 'triangle.gif';
  goptions colors=(black) cback=white
           device=imggif gsfname=grafout gsfmode=replace;

   /*-----------------------------------------------------------------*/
   /* Create the final annotate data set. Set XSYS and YSYS as          */
   /* appropriate for percentage computation. We want big bold numbers, */
   /* so we use the font SWISSB and set the size to 5.                      */
   /*-----------------------------------------------------------------*/

  data temp; set temp border;
```

```
     retain xsys '5' ysys '5' size 5 style 'swissb';
     if _n_=1 then set xxx;
     x=(x/maxx)*100; y=(y/maxy)*100;
     output;
run;

 /*-----Run PROC GANNO to create the desired image-------------------------*/
proc ganno anno=temp; run;
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Rick Langston
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000

Rick.Langston@sas.com