SESUG 2019 – Paper AT-110

The Knight's Tour in 3-Dimensional Chess

John R Gerlach; Navitas Life Sciences, Ltd.
Scott M. Gerlach; Dartmouth College

ABSTRACT

Three dimensional chess typically uses three chess boards such that a chess piece can traverse the several boards according to the rules for that piece. For example, the knight can remain on the board where it resides or move to another successive board, then move in a perpendicular fashion. In three-dimensional chess, the Knight's Tour is a sequence of moves on multiple 8x8 chess boards such that the knight visits each square only once. Thus, for three boards, there would be 192 squares visited only once. The paper, *The Knight's Tour in Chess – Implementing a Heuristic Solution* (Gerlach 2015), explains a SAS® solution for finding such tours on a single chess board, starting from any square. This paper discusses several scenarios and SAS solutions for generating the Knight's Tour using multiple chess boards.

INTRODUCTION

The Knight's Tour in three-dimensional (3-D) chess requires an understanding of how the knight-piece moves from one board to another. While remaining on the same chess board, the knight moves in its traditional L-shaped manner: two-steps in one direction, then one step in another direction; otherwise, one step in one direction, then two steps in another direction. However, in 3-D chess the knight moves a bit differently when it moves to another board. Assuming that the knight sits on the lowest board, the knight can move upward to the next board, then move two steps in a perpendicular direction, staying on the board, of course. Similarly, the knight can move upward two boards, then move one step, again in a perpendicular fashion. Obviously, the initial step might move downward accordingly, depending on whether the knight started on the top or middle board.

As explained in the author's previous paper, *The Knight's Tour in Chess – Implementing a Heuristic Solution* (Gerlach 2015), the solution to this interesting problem is premised on a simple heuristic rule first proposed by the German mathematician H.C. Warnsdorff. The heuristic rule states:

*Always move the knight to an adjacent, unvisited square with **minimal** degree.*

The term "minimal degree" indicates *the minimum number of **unvisited** available squares*.

This heuristic approach requires knowing how many moves the knight can make from any position on the chess board. This information is stored in the data set KNIGHTMOVES and used as a 2-dimensional matrix, which is maintained, decremented accordingly for each move, as the knight attempts to complete its tour. Table 1 shows the initialized matrix using the notation common in chess (i.e. Positions ***a8*** through ***h1***). Obviously, the center of the board (e.g. Position ***d5***) offers the most possible moves while Position ***a8*** has only two possible moves (***b6***, ***c7***). It is the combination of the KNIGHTMOVES data set and the heuristic rule that solves the Knight's Tour problem.

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***
8	2	3	4	4	4	4	3	2
7	3	4	6	6	6	6	4	3
6	4	6	8	8	8	8	6	4
5	4	6	8	***8***	8	8	6	4
4	4	6	8	8	8	8	6	4
3	4	6	8	8	8	8	6	4
2	3	4	6	6	6	6	4	3
1	2	3	4	4	4	4	3	2

Table1. Listing of possible knight moves for a single chess board.

Warnsdorff's rule does not guarantee a solution; however, the proposed SAS solution explained in the aforementioned paper generates 64 Knight's Tours on a single 8x8 (hence, 2-dimensional) chess board – with a bit of tweaking by its author. This paper discusses several solutions that includes a **third** dimension, that is, determining the Knight's Tour using three standard chess boards.

SOLUTION #1 – A SIMPLE EXTENSION

The first proposed solution is actually a simple extension of the so-called 2-dimensional problem, that is, using Warnsdorff's rule on a single chess board. Given that the knight's tour has been solved already from any starting position on a standard 8x8 chess board, consider the following idea:

Determine the knight's tour for each board ***independently, in succession,***
using the single board solution.

Simple, right? But how do you discern the next position on the adjacent board? Upon completion of the knight's tour on Board #1, simply move the knight one level upward according to the rules of 3-dimensional chess, that is, moving two squares in perpendicular fashion, ascertain where the knight is located on the successive board, then proceed using the single board solution. Wherever the knight's tour ends on a successive board, the next tour is obtained using the same method. Keep in mind that the location of the knight on the successive chess board is NOT the coordinate of the last step in the just completed tour; rather, it is the location of the legal 3-D chess move on the successive chess board. For example, Table 2 displays the knight's tour for two boards illustrating how the knight moved from the last step in the first tour, Position ***b2*** on Board #1, to Position ***b4*** on Board #2, becoming Step # 65, then continuing with the tour all the way to Step #128, Position ***a6***. **Note:** Knight Tours in this paper include *first and last steps* *italicized in red* in order to facilitate reading the tour.

Board #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	***1***	16	31	34	3	18	21	50
7	30	35	2	17	32	49	4	19
6	15	44	33	60	41	20	51	22
5	36	29	42	45	54	59	48	5
4	43	14	61	40	47	52	23	58
3	28	37	46	53	62	55	6	9
2	13	***64***	39	26	11	8	57	24
1	38	27	12	63	56	25	10	7

Board #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	84	127	80	95	102	99	78	97
7	81	94	83	124	79	96	105	100
6	***128***	85	126	103	116	101	98	77
5	93	82	123	90	125	104	111	106
4	86	***65***	92	115	110	117	76	119
3	71	68	89	122	91	120	107	112
2	66	87	70	73	114	109	118	75
1	69	72	67	88	121	74	113	108

Table 2. Display of two Knight Tours generated by Solution #1.

Solution #1 consists of two macros: **%ktour**, which generates the Knight's Tour; and, **%nxtcoord**, which discerns an appropriate coordinate where a new tour begins on the successive board. The **%ktour** macro has three parameters defining the *i*th board, along with the row-column starting position, which is initially arbitrary, then determined by the rules of 3-D chess. Because this macro is used repeatedly for successive tours, it is necessary to compute the appropriate Start / End position. For example, the first tour will have position values ranging from 1 to 64; whereas, the third tour will have position values ranging from 129 to 192. The Data Null step inside the **%ktour** macro accomplishes the task while a subsequent Data step generates the actual tour.

Although the **%ktour** may seem intricate, there are only three tasks being performed.

1. Assign the BOARD matrix by finding a legal "best" move based on the heuristic rule.
2. Update the MOVES matrix, decrementing by 1, the number of moves available based on the latest position.
3. Display the Knight's Tour.

```
%macro ktour(board,row,col);
 %* Create macro variables denoting the START / END of a tour ;;
    data _null_;
       end   = &board.*64;
       start = (end - 64) + 1;
       call symput('start',left(put(start,8.)));
       call symput('end',  left(put(end,8.)));
    run;
%* Determine a Knight's Tour ;;
    data board&board.;
       retain r &row. c &col. &svars. &mvars.;
       array board{8,8} &svars.;     ← S11, S12, . . ., S88 ;
       array moves{8,8} &mvars.;     ← M11, M12, . . ., M88 ;
       set knightmoves;
       board{r,c} = &start.;
       do position = (&start.+1) to &end.;
          nxtr   = 0;
          nxtc   = 0;
          pmoves = 9;    ← Highest possible number of moves ;

  %* Determine the best possible move per the knight's standard moves ;;
          do step1 = -2,-1,1,2;
             do step2 = -2,-1,1,2;
                if (abs(step1) ne abs(step2))
                   then do;
                      if 1 le (r+step1) le 8 and 1 le (c+step2) le 8
                         and board(r+step1,c+step2) eq .
```

```
                              then do;
                                 if moves{r+step1,c+step2} lt pmoves
                                    then do;
                                       nxtr   = r+step1;
                                       nxtc   = c+step2;
                                       pmoves = moves{r+step1,c+step2};
                                       end;
                                 end;
                           end;
                     end;
                  end;
   %* Assign the position assuming it is legal ;;
               if 1 le nxtr le 8 and 1 le nxtc le 8
                  then do;
                     r = nxtr;
                     c = nxtc;
                     board{r,c} = position;
   %* Update the MOVES matrix, decrementing by 1 ;;
                     do step1 = -2,-1,1,2;
                        do step2 = -2,-1,1,2;
                           if (abs(step1) ne abs(step2))
                              and 1 le (r+step1) le 8 and 1 le (c+step2) le 8
                                 then moves{r+step1,c+step2} =
                                    moves{r+step1,c+step2}-1;
                           end;
                        end;
                     end;
               end;
            keep &svars. &mvars.;
         run;
  %* Display the Knight's Tour ;;
         %showBoard(dsn      = board&board.,
                    elements = &svars.,
                    title    = Knights Tour on Board #&board.);
  %mend ktour;
```

The **%ktour** macro is actually a rehash of the original SAS solution; whereas, the second macro **%nxtcoord** is the crux of Solution #1, that is, the "Simple Extension" component. Upon completion of the first tour, the data set BOARD1 contains the knight's tour, stored as a two-dimensional matrix. The macro easily locates the last step by traversing the matrix. Then, adhering to the rules of 3-D chess, the next location is determined. Recall that the knight moves only one level, in succession, thereby moving two squares in a perpendicular fashion. If the move is legal (i.e. the knight stays on the board), the coordinate is stored in the global macro variables **R** and **C**, and the Data Null step terminates.

```
  %macro nxtcoord(board);
     data _null_;
        array board{8,8} &svars.;
        set board&board.(keep=s:);
  %* Find the location of the last step in the tour ;;
        do j = 1 to 8;
           do k = 1 to 8;
              if board{j,k} eq max(of board{*})
                 then do;  cur_row=j; cur_col=k; leave; end;
              end;
           end;
```

```
%* Find the location of the "First" step on the Successive board ;;
      do step1 = -2,2;
         do step2 = -2,2;
            if 1 le (cur_row+step1) le 8
               then do;
                  nxt_row = cur_row + step1;
                  nxt_col = cur_col;
                  end;
               else if 1 le (cur_col+step2) le 8
                  then do;
                     nxt_row = cur_row;
                     nxt_col = cur_col + step2;
                     end;
            if (cur_row ne nxt_row) or (cur_col ne nxt_col)
               then leave;
            end;
         end;
%* Store the coordinates in macro variables used for the %ktour macro ;;
      call symput('r', trim(left(put(nxt_row,2.))));
      call symput('c', trim(left(put(nxt_col,2.))));
   run;
%mend nxtcoord;
```

Table 3 shows how Solution #1 generates the Knight's Tour for three chess boards. The %knightmoves macro executes only once, generating the crucial KNIGHTMOVES data set while the %ktour and %nxtcoord macros formulate the three tours. Notice that the initial tour begins at Board #1, Position ***a8***, which is arbitrary. Actually, the first tour can begin at any position. Also, it is noteworthy that this solution can proceed *ad infinitum*.

```
%knightmoves;

%ktour(1,1,1);          %nxtcoord(1);
%ktour(2,&r.,&c.);      %nxtcoord(2);
%ktour(3,&r.,&c.);

%ktour(1,5,4);          %nxtcoord(1);
%ktour(2,&r.,&c.);      %nxtcoord(2);
%ktour(3,&r.,&c.);      %nxtcoord(3);
     :     :     :      :     :
```

Table 3. Generating the Knight's Tours using Solution #1.

SOLUTION #2 – THE "GLUE" METHOD

Another proposed solution, discovered during the research for this paper, is called the "Glue" method, which requires a *closed* tour, that is, the start and end positions of the tour are one knight move away. Table 4 displays two Knight's Tours using Warnsdorff's method, a closed tour where Steps #1 and #64 are one move from each other and an open tour where Steps #1 and #64 are not, yet still a valid tour.

Closed Tour

	a	b	c	d	e	f	g	h
8	20	5	38	47	22	7	26	45
7	37	50	21	6	39	46	23	8
6	4	19	56	51	48	25	44	27
5	55	36	49	32	57	40	9	24
4	18	3	54	61	52	43	28	41
3	35	***64***	33	58	31	60	13	10
2	2	17	62	53	12	15	42	29
1	63	34	***1***	16	59	30	11	14

Open Tour

	a	b	c	d	e	f	g	h
8	42	25	22	7	48	35	20	5
7	23	8	41	36	21	6	47	34
6	26	43	24	49	46	57	4	19
5	9	40	45	56	37	50	33	58
4	44	27	52	39	62	59	18	3
3	13	10	55	60	51	38	63	32
2	28	53	12	15	30	61	2	17
1	11	14	29	54	***1***	16	31	***64***

Table 4. A closed knight's tour and an open knight's tour.

This method seems more like cheating because it does little more than clone an existing closed tour: it does not truly generate a knight's tour. Instead, the process recodes the numerical sequence (i.e. steps) in the tour based on the last step and the subsequent legal move to the next board. So, given a closed tour, how does it recode the sequence of values? First of all, it must discern a legal move with respect to 3-D chess. For example, as shown in Table 5, the knight can move from Board #1, Position ***b3***, to Board #2, Position ***d3***, which denotes the knight's move on the next board, followed by two squares in perpendicular fashion. There are other possible moves, but for this discussion, the point is to illustrate the cloning process.

As shown in Table 5, Position ***d3*** on Board #1 denotes Step #58 (the array element **board**{6,4}), which becomes the ***pivotal value*** for the cloning process. Because of an inherent property of a closed tour, it is possible to recode the Step values by computing two incremental values using the following formulas:

- **INCR1 = MAX(of board{*}) – board{6,4} + 1 = 64 – 58 = 7**
- **INCR2 = MAX(of board{*}) – board{6,4} + 65 = 6 + 65 = 71**

Board #1: Initial (Closed) Tour

	a	b	c	d	e	f	g	h
8	20	5	38	47	22	7	26	45
7	37	50	21	6	39	46	23	8
6	4	19	56	51	48	25	44	27
5	55	36	49	32	57	40	9	24
4	18	3	54	61	52	43	28	41
3	35	***64***	33	58	31	60	13	10
2	2	17	62	53	12	15	42	29
1	63	34	***1***	16	59	30	11	14

Board #2: Successive Tour

	a	b	c	d	e	f	g	h
8	91	76	109	118	93	78	97	116
7	108	121	92	77	110	117	94	79
6	75	90	127	122	119	96	115	98
5	126	107	120	103	***128***	111	80	95
4	89	74	125	68	123	114	99	112
3	106	71	104	***65***	102	67	84	81
2	73	88	69	124	83	86	113	100
1	70	105	72	87	66	101	82	85

Table 5. An initial closed knight's tour and its successor beginning at a proper starting position.

Following the example in Table 5, the variables INCR1 and INCR2 will have the values 7 and 71, respectively. Then, simply traverse the first tour (matrix) and increment accordingly, that is, pivoting on the value 58. Thus, steps ranging from 58 to 64 are incremented by 7, thereby ranging from 65 (which is the first step on the successive board) to 71; whereas, those steps below Step #58 are incremented by 71. Thus, Steps #1 through #57 will range from 72 to 128, as shown in Example #1 of Table 6. And, it works!

Even more ironic concerning this method – Given a closed tour, Examples 2 and 3 in Table 6 illustrate that *any* position can be selected as the pivotal position for the recoding process, albeit not following the rules for 3-D chess. Also, notice that the result is always another closed tour.

Although the implementation emulates the Solution #1 using the KNIGHTMOVES data set and the heuristic rule, it is necessary to designate the starting position (row, column) knowing that the result will be a closed tour.   Then, the ***%glue*** macro generates the subsequent tour by transforming the values of the first (closed) tour into a new Knight's Tour.

Board #1: (Closed) Tour

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	20	5	38	47	22	7	26	45
7	37	50	21	6	39	46	23	8
6	4	19	56	51	48	25	44	27
5	55	36	49	32	57	40	9	24
4	18	3	54	61	52	43	28	41
3	35	***64***	33	58	31	60	13	10
2	2	17	62	53	12	15	42	29
1	63	34	***1***	16	59	30	11	14

Board #2: Example #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	91	76	109	118	93	78	97	116
7	108	121	92	77	110	117	94	79
6	75	90	127	122	119	96	115	98
5	126	107	120	103	***128***	111	80	95
4	89	74	125	68	123	114	99	112
3	106	71	104	***65***	102	67	84	81
2	73	88	69	124	83	86	113	100
1	70	105	72	87	66	101	82	85

Board #2: Example #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	89	74	107	116	91	76	95	114
7	106	119	90	75	108	115	92	77
6	73	88	125	120	117	94	113	96
5	124	105	118	101	126	109	78	93
4	87	72	123	66	121	112	97	112
3	104	69	102	127	100	***65***	82	79
2	71	86	67	122	81	84	111	98
1	68	103	70	85	***128***	99	80	83

Board #2, Example #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	104	89	122	67	106	91	110	***65***
7	121	70	105	90	123	66	107	92
6	88	103	76	71	68	109	***128***	111
5	75	120	69	116	77	124	93	108
4	102	87	74	81	72	127	112	125
3	119	84	117	78	115	80	97	94
2	86	101	82	73	96	99	126	113
1	83	118	85	100	79	114	95	98

Table 6. A closed knight's tour with three successive closed tours.

The ***%glue*** macro, shown below, obtains the Step-value of the knight's next position, generates a new tour using the "Glue" method, then displays the new tour.

```
%macro glue(board,r,c);
%* Obtain Step-value of the knight's next position ;;
   data _null_;
      retain r &r. c &c.;
      array board{8,8} s11-s18 s21-s28 . . .  s81-s88
      set board%eval(&board.-1);
      call symput('pval',trim(left(put(board{r,c},3.))));
   run;

%* Generate new tour by "Glue" method ;;
   data board%eval(&board.+1);
      retain &svars. &mvars. pval &pval. incr1 incr2;
      array board{8,8} &svars.;
      set board%eval(&board.-1);      ← Process previous tour.
      if _n_ eq 1
         then do;    ← L.T. and G.E. Pivotal value
            incr1 = max(of board{*}) - board{&r.,&c.} + 1;
            incr2 = max(of board{*}) - board{&r.,&c.} + 65;
            end;
      do r = 1 to 8;         ← Recode board, accordingly.
         do c = 1 to 8;
            if board{r,c} ge &pval.
```

```
                then board{r,c} = board{r,c} + incr1;
                else board{r,c} = board{r,c} + incr2;
             end;
          end;
       drop r c incr1 incr2 pval;
    run;
 %* Display the new tour ;;
    %ShowBoard(dsn      = board%eval&board.,
               elements = &svars.,
               title    = Knights Tour on Board #%eval(&board.+1));
 %mend glue;
```

Table 8 shows a closed tour beginning on Board #1, Position ***b1***. As an exercise for the reader, formulate the successive tour (i.e. Board #2) using the following criteria: **Pivotal Value=60, INCR1=5, and INCR2=69**. Keep in mind that the Pivotal Value was determined by the rules of 3-D chess; whereas, the incremental variables were computed using formulas.

Board #1: (Initial Closed Tour)

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***
8	25	22	5	34	27	12	7	10
7	4	35	26	23	6	9	28	13
6	21	24	45	36	33	30	11	8
5	44	3	58	31	50	37	14	29
4	59	20	51	46	63	32	49	38
3	2	43	62	57	52	47	54	15
2	19	60	41	***64***	17	56	39	48
1	42	***1***	18	61	40	53	16	55

Board #2 (Next Tour)

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***
8	94	91	74	103	96	81	76	79
7	73	104	95	92	75	78	97	82
6	90	93	114	105	102	99	80	77
5	113	72	127	100	119	106	83	98
4	***128***	89	120	115	68	101	118	107
3	71	112	67	126	121	116	123	84
2	88	***65***	110	69	86	125	108	117
1	111	70	87	66	109	122	85	124

Table 7. Board #2 created from Board #1 using Step #60 (Position ***b2***) as the pivotal value.

SOLUTION #3 – TRAVERSING THE BOARDS DURING THE TOUR

The first two solutions produce valid tours, albeit in a limited way. The first solution merely extended Warnsdorff's heuristic rule by generating tours independently for each board, then moving the knight to a successive board according the rules of 3-D chess and finding the next tour. The second solution, the so-called "Glue" method, is a scheme to recode a given closed tour in order to produce a new tour. It does not generate a Knight's Tour from scratch. The "Glue" method is little more than an isomorphism – and a sham. Neither solution generates the Knight's Tour as one might imagine, that is, ***traversing several chess boards during the tour***.

The proposed solution takes Warnsdoff's heuristic rule to new heights. The idea is the same: creating and updating the KNIGHTMOVES data set in tandem with the heuristic rule for discerning the next knight move. However, this time the number of possible moves represents three chess boards, not just one. In this situation, it is necessary to know *a priori* the number of possible knight moves **from any of 192 possible squares** (i.e. 3 chess boards each having 64 squares), because now there is a third factor called the *Level*: Board #1, #2, and #3.

Table 8 displays the three boards in juxtaposition and includes several examples of how the knight moves from its initial position to all possible moves. The coding convention clearly indicates rows *1* through *8* and columns *a* through *h* that facilitates enumerating the possible moves. Keep in mind the rules for 3-D chess.

Board #1

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Board #2

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

Board #3

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

	# Possible	---------------------- *To* Position	----------------------	
From **Position**	**Moves**	**Board #1**	**Board #2**	**Board #3**
Board #1 / Position a8	6	b6, c7	a6, c8	a7, b8
Board #2 / Position d5	16	b5, f5, d7, d3	b4, b6, c3, c7, e3, e7, f4, f6	b5, f5, d7, d3
Board #3 / Position b3	10	b2, b4, c3	b1, b5, d3	a1, a5, c1, c5, d2, d4

Table 8. Listing of possible knight moves.

Consider the following important points before proceeding:

1. The Knight's Tour is displayed as *a sequence of integers*, ranging from 1 to 192. Theoretically, for example, Step #1 begins at Board #1, Position *a8* and Step #192 lands on Board #3, Position *e1*.
2. Using three boards, the knight's *second* step depends on its initial step, that is, whether the piece moves *one* or *two* levels will depend on its initial location. For example, the knight cannot jump two levels if it resides initially on the middle chess board, since there are only three boards.
3. The number of possible moves for the knight ranges from 6 to 16 (See Table 9), which is significantly more than the Knight's Tour problem using a single board.
4. When moving to another board, the knight lands on the *same-named* square (e.g. *a8* on one board moves to *a8* onto another board) before moving to its final destination, accordingly.
5. If the initial move begins from Board #2, the knight can move only one level.
6. Obviously, the move must be legal and the knight remains on the board.

The first task is to calculate the number of possible knight moves from *any square* on *any board*, as shown in Table 9. This information would be stored in a data set called KNIGHTMOVES, which becomes the only input to the proposed SAS solution, implemented as a 3-dimensinal array, called the MOVES matrix. The knight begins its tour from a given starting position, seeks the most reasonable next move, and then updates the MOVES matrix, by decrementing by 1 those places where the knight could go from the new position.

The objective is to create a data set that contains 1 observation having 192 variables whose naming convention intuitively indicates a 3-dimensional array denoting the LEVEL, ROW, and COLUMN. These *m*-variables (*m* denotes MOVES) contain the number of possible moves from all 192 positions.

The abridged Data step below uses three DO-loops to address each element in the 3-dimensional - *moves{level,r,c}*.

In order to count the number of possible knight moves, we first focus on the board where the knight resides; thereby considering the traditional L-shaped movements, which are implemented using two DO-loops and their respective loop control variables: STEP1 and STEP2. Given that the step is legal, the COUNTER variable is incremented. Then, in order to count the possible moves onto the other boards, *from the same location*, there are only two scenarios:

1. The knight jumps to the next board, then moves TWO squares in a perpendicular fashion.
2. The knight jumps to the second board, then moves ONE square in perpendicular fashion.

If the knight resides on level 1 or 3, then both scenarios prevail; however, if the knight resides on level 2, then only the first scenario prevails. Of course, the move must be legal in order to increment the COUNTER.

```
data knightmoves;     One observation, keeping only the M-variables.
   array moves{3,8,8}  m111-m118 . . m188 m211-m218 . . m288
                       m311-m318 . . m388;
   do level = 1 to 3;     Process all three 8x8 boards.
      do r = 1 to 8;
         do c = 1 to 8;   For a given knight position {r,c}.
            counter=0;    Initialize the counter to zero.

 Knight moves in L-shaped fashion. If legal move then increment counter.
            select(level);     Knight moves one level or two levels.
               when(1,3)       If legal move then increment counter.
               when(2)         If legal move then increment counter.
               end;
            moves{level,r,c} = counter;    Assign # possible moves.
      end; end; end;
run;
```

Board #1

	a	b	c	d	e	f	g	h
8	6	8	10	10	10	10	8	6
7	8	10	13	13	13	13	10	8
6	10	13	16	16	16	16	13	10
5	10	13	16	16	16	16	13	10
4	10	13	16	16	16	16	13	10
3	10	13	16	16	16	16	13	10
2	8	10	13	13	13	13	10	8
1	6	8	10	10	10	10	8	6

Board #2

	a	b	c	d	e	f	g	h
8	6	7	10	10	10	10	7	6
7	7	8	12	12	12	12	8	7
6	10	12	16	16	16	16	12	10
5	10	12	16	16	16	16	12	10
4	10	12	16	16	16	16	12	10
3	10	12	16	16	16	16	12	10
2	7	8	12	12	12	12	8	7
1	6	7	10	10	10	10	7	6

Board #3

	a	b	c	d	e	f	g	h
8	6	8	10	10	10	10	8	6
7	8	10	13	13	13	13	10	8
6	10	13	16	16	16	16	13	10
5	10	13	16	16	16	16	13	10
4	10	13	16	16	16	16	13	10
3	10	13	16	16	16	16	13	10
2	8	10	13	13	13	13	10	8
1	6	8	10	10	10	10	8	6

Table 9. The KNIGHTMOVES metadata for three chess boards. Notice the symmetry.

Solution #3 is implemented as a single macro with no parameters. It is designed to traverse all 192 squares in search of the Knight's Tour, starting from each position. This lengthy macro follows the same approach of utilizing the KNIGHTMOVES data set and moving the knight on the board where it resides, then considering the other levels, accordingly. Thus, for a given starting position (Level, Row, Column), a Data step attempts to generate the Knight's Tour by initializing the BOARD 3-dimensional matrix with the value 1 (denoting Step #1) then traversing the matrix to determine the position of Step #2, and so on. The Data step considers the L-shaped moves on the board where the knight resides, obtaining the "minimal degree" in accordance to the heuristic rule, then considers the 3-D chess moves to the other boards, likewise. Depending on the level where the knight resides will determine the subsequent checks in search of the "minimal degree."

The following Data _null_ step determines whether the attempt to formulate the Knight's Tour was successful. If so, then a 3-D version of the **%showboard** macro generates the report. Table 10 shows an example of a successful and a failed tour, the latter tour getting stuck at Step #189.

```
data _null_;
   array board{3,8,8} s111-s118 … s181-s188 … s311-s318 … s381-s388;
   set solution;
   call symput('solved',left(put(n(of board{*}) eq 192,1.)));
run;
```

Successful Knight's Tour

Board #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	24	29	26	21	46	35	40	43
7	31	22	81	74	83	106	47	38
6	80	73	100	105	180	157	118	107
5	69	104	141	156	183	136	181	48
4	18	101	146	173	***192***	185	158	119
3	103	68	153	184	159	182	135	86
2	10	17	***128***	145	174	177	62	91
1	38	27	12	63	56	25	10	7

BOARD #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	27	20	55	34	41	50	109	36
7	54	33	76	115	110	37	84	49
6	19	96	111	78	99	114	139	120
5	32	77	148	113	140	163	58	85
4	95	112	127	144	179	176	123	138
3	14	7	142	149	162	137	160	59
2	***1***	94	15	178	143	150	175	122
1	6	13	2	93	66	161	60	87

BOARD #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	30	25	28	51	56	45	42	39
7	23	52	71	82	75	116	57	44
6	70	79	98	155	132	165	108	117
5	53	72	131	166	187	190	133	165
4	102	97	154	189	170	167	186	121
3	5	130	147	172	191	188	169	134
2	16	11	126	151	168	171	124	63
1	9	4	***129***	12	125	88	***65***	90

Failed Knight's Tour

Board #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	23	16	41	32	91	72	***65***	70
7	18	31	90	83	88	77	92	63
6	13	40	109	102	115	136	87	74
5	30	101	114	137	.	174	***128***	135
4	35	108	103	177	139	.	161	86
3	100	55	138	186	185	178	175	***129***
2	7	52	107	104	176	162	145	154
1	2	9	6	121	146	157	148	143

BOARD #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	14	33	22	27	60	69	44	73
7	21	26	59	78	45	84	61	68
6	34	15	46	111	96	79	126	85
5	25	58	97	164	127	152	173	62
4	12	51	110	119	170	163	132	153
3	57	38	123	98	165	168	151	142
2	36	11	118	169	140	131	166	133
1	5	56	37	10	167	150	141	130

BOARD #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	17	28	47	82	43	76	71	***64***
7	24	19	42	89	94	81	66	75
6	29	48	95	116	125	112	93	80
5	20	39	124	113	183	188	160	67
4	49	54	117	187	.	179	182	134
3	4	99	106	180	***189***	184	172	159
2	***1***	50	53	120	171	181	155	144
1	8	3	122	105	156	147	158	149

Table 10. A successful tour and a failed tour (stopping at step #189).

FOUR CHESS BOARDS

Warnsdorff heuristic rule works for a single chess board and three chess boards, the latter adhering to 3-D chess rules. What about four chess boards? Is Warnsdorff's heuristic rule strong enough? It turns out – Yes! Not surprisingly, however, the 4-board solution requires a fresh study of how the knight moves between 4-boards. For example, if the knight sits on Board #1, it cannot legally jump to Board #4; that is, it must first jump to either Board #2 or Board #3 during the tour in order to get to Board #4, which must be included as part of the solution. Hence, one might think that this hurdle would make the heuristic rule too weak or biased. Surprisingly, the rule was able to generate 38 tours out of a possible 256 tours; only a 46.9% success rate, rather impressive. Also, adding a fourth level affects the logic concerning the creation of the KNIGHTMOVES data set. In this case, all four levels must consider perpendicular moves both one step and two steps, unlike the 3-board problem. The code below highlights the portion where the knight is moving to another board to discern a legal move in order to increment the COUNTER variable.

```
data knightmoves
   array moves{4,8,8} %gen_vars(m,4);
   do level = 1 to 4;
      do r = 1 to 8;
         do c = 1 to 8;
            counter=0;

   Consider L-shaped moves where the knight resides (not shown).
   Consider knight moves to the other levels using SELECT/WHEN.

            select(level);
               when(1,2,3,4) do;
                  do step2 = -2,2;
                     if 1 le (r+step2) le 8  then counter+1;
                     if 1 le (c+step2) le 8  then counter+1;
                     end;
                  do step2 = -1,1;
                     if 1 le (r+step2) le 8  then counter+1;
                     if 1 le (c+step2) le 8  then counter+1;
                     end;
                  end;
               otherwise;
               end;
            moves{level,r,c} = counter;
            end;
         end;
      end;
   keep m:;
run;
```

It is interesting to note that the MOVES matrix for 4-boards contains the same values for each level, as shown in Table 11, which makes sense because each level manifests the same movements on their respective boards, as well as to other appropriate boards. For example, the value of the array element moves{n,3,4} (Position ***d4*** in chess notation) indicates 16 possible moves regardless of which level the knight resides initially.

Boards #1 and #4

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	6	8	10	10	10	10	8	6
7	7	10	13	13	13	13	10	9
6	9	13	16	16	16	16	13	11
5	9	13	16	16	16	16	13	11
4	9	13	16	16	16	16	13	11
3	9	13	16	16	16	16	13	11
2	7	10	13	13	13	13	10	9
1	6	8	10	10	10	10	8	6

Boards #2 and #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	8	10	13	13	13	13	10	8
7	10	12	16	16	16	16	12	10
6	13	16	20	20	20	20	16	13
5	13	16	20	20	20	20	16	13
4	13	16	20	20	20	20	16	13
3	13	16	20	20	20	20	16	13
2	10	12	16	16	16	16	12	10
1	8	10	13	13	13	13	10	8

Table 11. Possible knight moves for four boards.

The solution to the 4-board problem consists of a lengthy Data step (not shown) because of all the steps the knight might consider during the tour. The method is the same as Solution #3, just more involved. The Knight's Tour, shown in **Table 12**, begins at Board #1, Position ***e8***. As mentioned previously, the Steps: 1, 64, 65, 128, 129, 192, 193, 256 are highlighted in red in order to facilitate reading the tour. Notice that the knight's movement traverses all four boards, almost immediately. For example, Step #2 to Step #3: the knight moves from Board #2 to Board #4. Although the integer values indicate higher values in Board #4, the knight's tour shown below is valid.

Board #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	81	88	85	118	***1***	10	13	6
7	86	95	124	109	120	117	22	11
6	125	108	201	160	169	158	73	20
5	176	131	178	185	200	151	144	23
4	91	196	199	202	179	172	39	72
3	130	175	184	197	194	143	68	31
2	55	52	195	138	171	40	71	38
1	50	57	104	43	140	135	***64***	29

Board #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	84	77	80	111	98	5	2	9
7	79	110	121	116	155	150	99	4
6	92	161	154	157	148	115	18	25
5	***129***	***192***	183	166	153	156	149	100
4	162	107	164	173	170	147	114	19
3	49	132	***193***	182	165	152	101	28
2	60	163	106	133	102	137	34	41
1	47	44	61	136	105	42	67	32

Board #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	87	82	97	76	15	112	7	12
7	96	89	94	119	122	75	14	21
6	83	126	123	168	159	146	113	8
5	90	177	204	191	180	167	74	27
4	59	174	189	186	205	142	145	24
3	54	187	198	203	190	181	36	69
2	51	58	103	188	141	134	63	30
1	56	53	46	139	62	***65***	70	37

Board #4

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	78	127	238	211	224	229	16	3
7	93	212	223	228	251	210	225	26
6	***128***	239	250	237	230	227	242	17
5	213	222	231	254	241	252	209	226
4	232	249	240	247	236	255	206	243
3	221	214	235	***256***	253	246	217	208
2	48	233	248	219	216	207	244	33
1	45	220	215	234	245	218	35	66

Table 12. The Knight's Tour using four chess boards. Note the step highlighted in red.

USEFUL UTILITIES – SUPPLEMENTAL

There are several utilities that facilitate the SAS solution for the Knight's Tour. One such utility enumerates the variables representing the 3-dimensional array, rather than listing all 192 variables (e.g. s111 . . . s118). Notice the following intuitive naming convention for the MOVES and SOLUTION variables:

<M **|** S><Level><Row><Column> (e.g. m238, s341)

where M and S denote the Moves and Solution 3-dimensional matrices. Level denotes the board in tier fashion. Row and Column indicate the coordinate on a given board (level). Thus, the element m238 denotes the number of knight moves *from* Board #2, Position ***h6*** (row 3, column 8); whereas, the array element s341 denotes the *n*th move *at* position Board #3, Position ***a5*** (row 4, column 1). The following utility generates these variables easily. Obviously, this utility can be used for more than three chess boards.

```
%macro gen_vars(type, levels);
   %do level = 1 %to &levels.;
      %do row = 1 %to 8;
         %do column = 1 %to 8;
            &type.&level.&row.&column.
            %end;
         %end;
      %end;
%mend gen_vars;
```

The %Show_Boards macro, another invaluable utility, generates a report showing the three chess boards in vertical juxtaposition. The BOARD matrix can represent either the Moves or Solution matrix as defined by the elements parameter. Keep in mind that these variables actually reside in a single observation data set, each consisting of 192 variables. The utility converts the matrix into a readable format listing row and columns for each board level. It is a good exercise to enhance the macro to handle *N* chess boards.

```
%macro Show_Boards(dsn      = solution,
                   elements = %gen_vars(s,3),
                   title    = 3-D Chess Board);
   data board;
      array board{3,8,8} &elements.;
      array cols{8} c1-c8;
      set &dsn.;
      do level = 1 to 3;
         do r = 1 to 8;
            do c = 1 to 8;
               cols{c} = board{level,r,c};
               end;
            output;
            end;
          end;
      keep level c1-c8;
   run;
   proc report data=board nowindows headskip;
      columns level c1-c8;
      define level / order   width=5  'Level';
      define c1 / display width=3  '';   define c2 / display width=3 '';
      define c3 / display width=3  '';   define c4 / display width=3 '';
      define c5 / display width=3  '';   define c6 / display width=3 '';
      define c7 / display width=3  '';   define c8 / display width=3 '';
      break after level / skip;
      format c1-c8 best3.1;
      title2 "&title.";
   run;
%mend Show_Boards;

%Show_Boards(dsn=solution, title=Knight Tour);
```

CONCLUSION

The Knight's Tour problem has been around for centuries and has been investigated by famous mathematicians including Leonhard Euler. The heuristic rule proposed by the mathematician H.C. Warnsdorf has been very successful in generating numerous tours, even using more than three boards. The first two proposed solutions were mere extensions of the single board problem; whereas, the third solution represents a bona fide tour that traverses all the boards during a successful tour. The creation and maintenance of the KNIGHTMOVES data set, along with the intricate movement of the knight in search of "minimal degree" posed quite a challenge, especially when applying the heuristic rule to more than three chess boards. Does Warnsdorff's heuristic rule hold for five, six boards? Yes, it does. More than six? Probably.

REFERENCES

https://www.futilitycloset.com/2014/11/10/warnsdorffs-rule/

Gerlach, John R. *The Knight's Tour in Chess – Implementing a Heuristic Solution,* SAS Global Forum, 2015.

Keen, M.R. *The Knight's Tour.* http://www.markkeen.com/knight/index.html.

Kumar, Awani, et.al. *Non-crossing Knight's Tour n 3-Dimension*.

Neilan, Fredrick Scott. *(Knight)^3: A Graphical Perspective of the Knight's Tour on a Multi-Layered Chess Board,* Bridgewater State University, 2016.

Squirrel, Douglas; Cull, P*. (1996). A Warnsdorff-Rule Algorithm for Knight's Tours on Square Boards.*

CONTACT INFORMATION

Name:	John R. Gerlach	Scott M. Gerlach
Enterprise:	Navitas Life Sciences, Ltd.	Dartmouth College
E-mail:	gerlachj@dataceutics.com	Scott.M.Gerlach@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX A – KNIGHTMOVES DATA SET - SINGLE CHESS BOARD

```
data knightmoves;
   array board{8,8} m11-m18 m21-m28 m31-m38 m41-m48
                    m51-m58 m61-m68 m71-m78 m81-m88;
   do r = 1 to 8;
      do c = 1 to 8;
         counter=0;
         do step1 = -2,-1,1,2;
            do step2 = -2,-1,1,2;
               if (abs(step1) ne abs(step2))
                  then do;
                     if 1 le (r+step1) le 8 and 1 le (c+step2) le 8
                        then counter+1;
                     end;
               end;
            end;
         board{r,c} = counter;
         end;
      end;
   drop r c step1 step2 counter;
run;
```

APPENDIX B – TWO KNIGHT TOURS USING A SINGLE CHESS BOARD

Tour #1

	a	b	c	d	e	f	g	h
8	1	16	31	34	3	18	21	50
7	30	35	2	17	32	49	4	19
6	15	44	33	60	41	20	51	22
5	36	29	42	45	54	59	48	5
4	43	14	61	40	47	52	23	58
3	28	37	46	53	62	55	6	9
2	13	64	39	26	11	8	57	24
1	38	27	12	63	56	25	10	7

Tour #2

	a	b	c	d	e	f	g	h
8	20	33	16	1	26	31	14	39
7	17	2	19	32	15	38	27	30
6	34	21	42	25	36	29	40	13
5	3	18	35	54	41	50	37	28
4	22	43	24	49	62	55	12	51
3	7	4	59	46	53	48	63	56
2	44	23	6	9	58	61	52	11
1	5	8	45	60	47	10	57	64

APPENDIX C – TWO KNIGHT'S TOURS USING THREE CHESS BOARDS

SOLUTION #1

Start: Board #1, Position ***a8***
End: Board #4, Position ***e1***

BOARD #1

	a	b	c	d	e	f	g	h
8	1	16	31	34	3	18	21	50
7	30	35	2	17	32	49	4	19
6	15	44	33	60	41	20	51	22
5	36	29	42	45	54	59	48	5
4	43	14	61	40	47	52	23	58
3	28	37	46	53	62	55	6	9
2	13	64	39	26	11	8	57	24
1	38	27	12	63	56	25	10	7

BOARD #2

	a	b	c	d	e	f	g	h
8	84	127	80	95	102	99	78	97
7	81	94	83	124	79	96	105	100
6	128	85	126	103	116	101	98	77
5	93	82	123	90	125	104	111	106
4	86	65	92	115	110	117	76	119
3	71	68	89	122	91	120	107	112
2	66	87	70	73	114	109	118	75
1	69	72	67	88	121	74	113	108

BOARD #3

	a	b	c	d	e	f	g	h
8	139	156	141	164	137	154	161	176
7	142	165	138	155	162	175	136	153
6	157	140	163	168	181	160	177	174
5	166	143	182	159	178	187	152	135
4	129	158	167	186	169	180	173	190
3	144	183	146	179	188	191	134	151
2	147	130	185	170	149	132	189	172
1	184	145	148	131	192	171	150	133

SOLUTION #2 - "GLUE" METHOD

Start: Board #1, Positon ***a2***
End: Board #4, Postion ***b8***

BOARD #1

	a	b	c	d	e	f	g	h
8	19	62	15	30	37	34	13	32
7	16	29	18	59	14	31	40	35
6	63	20	61	38	51	36	33	12
5	28	17	58	25	60	39	46	41
4	21	64	27	50	45	52	11	54
3	6	3	24	57	26	55	42	47
2	1	22	5	8	49	44	53	10
1	4	7	2	23	56	9	48	43

BOARD #2

	a	b	c	d	e	f	g	h
8	126	105	122	73	80	77	120	75
7	123	72	125	102	121	74	83	78
6	106	127	104	81	94	79	76	119
5	71	124	101	68	103	82	89	84
4	128	107	70	93	88	95	118	97
3	113	110	67	100	69	98	85	90
2	108	65	112	115	92	87	96	117
1	111	114	109	66	99	116	91	86

BOARD #3

	a	b	c	d	e	f	g	h
8	149	192	145	160	167	164	143	162
7	146	159	148	189	144	161	170	165
6	129	150	191	168	181	166	163	142
5	158	147	188	155	190	169	176	171
4	151	130	157	180	175	182	141	184
3	136	133	154	187	156	185	172	177
2	131	152	135	138	179	174	183	140
1	134	137	132	153	186	139	178	173

APPENDIX D – TWO KNIGHT'S TOURS USING FOUR CHESS BOARDS

Start: Board #3, Position ***e3***
End: Board #4, Position ***d3***

BOARD #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	44	47	50	79	42	73	20	25
7	49	56	83	72	99	***64***	41	18
6	154	161	142	135	124	95	74	21
5	55	180	198	187	143	102	61	40
4	162	186	184	194	138	121	94	31
3	179	***193***	188	197	127	130	69	16
2	152	117	185	149	174	93	32	5
1	115	148	151	108	***129***	6	3	14

BOARD #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	51	78	43	46	59	38	29	22
7	54	157	98	77	82	71	60	39
6	141	134	125	166	97	76	37	30
5	158	200	156	133	126	131	70	17
4	153	170	165	182	175	96	75	36
3	114	181	199	171	132	103	88	67
2	163	172	183	176	145	120	35	10
1	112	109	146	119	104	89	68	7

BOARD #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	48	45	58	85	80	27	24	19
7	57	84	81	100	63	86	***65***	26
6	52	167	140	123	136	101	62	23
5	155	160	190	195	139	122	87	66
4	168	***192***	189	177	144	137	34	9
3	159	178	196	191	***1***	106	91	12
2	116	169	164	173	***128***	33	2	15
1	147	118	111	150	107	92	13	4

BOARD #4

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	53	211	208	203	224	235	206	28
7	209	202	225	242	207	204	223	236
6	212	243	210	227	234	251	240	205
5	201	226	255	250	241	228	237	222
4	244	213	246	233	254	239	252	229
3	215	218	249	***256***	247	232	221	238
2	113	245	214	217	220	253	230	8
1	110	216	219	248	231	105	90	11

Start: Board #4, Position ***h1***
End: Board #4, Position ***e3***

BOARD #1

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	66	71	74	59	44	51	40	33
7	73	68	103	98	101	58	35	38
6	90	97	130	135	122	113	52	41
5	85	134	187	158	139	108	57	26
4	96	157	172	147	186	123	112	47
3	163	188	185	196	177	140	25	20
2	166	171	178	189	146	115	14	11
1	87	164	195	176	141	12	19	4

BOARD #2

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	63	60	81	70	55	32	45	30
7	82	127	100	93	80	107	56	27
6	***65***	94	121	***128***	105	54	111	46
5	150	159	126	133	120	79	106	37
4	89	154	151	160	145	114	53	16
3	86	181	***192***	155	142	119	78	9
2	153	156	161	182	191	144	17	2
1	162	***193***	180	143	18	3	24	7

BOARD #3

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	72	69	62	75	50	43	34	39
7	67	92	83	102	99	76	49	36
6	84	***129***	104	131	136	109	42	29
5	91	132	149	138	125	118	77	48
4	152	95	198	173	148	137	110	21
3	167	170	179	184	197	124	117	6
2	88	183	190	169	174	13	22	15
1	165	168	175	194	23	116	5	10

BOARD #4

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
8	61	210	247	230	219	212	31	28
7	***64***	231	228	211	248	215	220	213
6	209	246	249	254	229	218	239	216
5	232	227	242	245	250	253	214	221
4	199	208	255	252	243	240	217	238
3	226	233	244	241	***256***	251	222	203
2	207	200	235	224	205	202	237	8
1	234	225	206	201	236	223	204	***1***