PharmaSUG 2015 – Paper BB01

The Knight's Tour in Chess – Implementing a Heuristic Solution

John R Gerlach, Cape Coral, FL

ABSTRACT

The Knight's Tour is a sequence of moves on a chess board such that a knight visits each square only once. Using a heuristic method, it is possible to find a complete path, beginning from any arbitrary square on the board and landing on the remaining squares only once. However, the implementation poses challenging programming problems. For example, it is necessary to discern viable knight moves, which changes throughout the tour. Even worse, the heuristic approach does not guarantee a solution. This paper explains a SAS® solution that finds a knight's tour beginning from every initial square on a chess board – Well, almost.

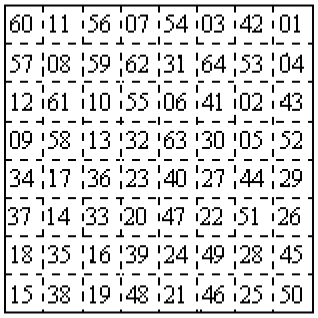
INTRODUCTION

The Knight's Tour is actually a mathematical (Hamiltonian Path) problem dating back centuries. Many solutions have been proposed, ranging from brute force algorithms to neural networks. This paper discusses a heuristic solution proposed in 1823 by the German mathematician H.C. Warnsdorff that states the following simple rule:

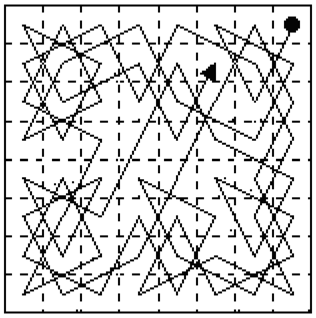
Always move the knight to an adjacent, unvisited square with minimal degree.

Starting from any square, the knight must move to an unvisited square that has the fewest successive moves. Choosing a square with the fewest successors avoids a possible dead-end when traversing the board. However, because Warnsdorff's rule is heuristic, it is not guaranteed to find a solution. Fortunately, the proposed SAS solution determines a complete path beginning from any square – Well, almost. Although there are numerous solutions for a given starting point, this solution attempts to generate sixty-four paths on a standard 8x8 chess board, one for each starting position.

Rather than show the knight's tour as a directional path, it is more expedient to display the ordinal, numerical sequence on the chess board, which more easily indicates the knight's moves. In the example below, the tour begins in the upper-right corner of the board and ends nearby, just one move away.



60	11	56	07	54	03	42	01
57	08	59	62	31	64	53	04
12	61	10	55	06	41	02	43
09	58	13	32	63	30	05	52
34	17	36	23	40	27	44	29
37	14	33	20	47	22	51	26
18	35	16	39	24	49	28	45
15	38	19	48	21	46	25	50



DISPLAYING THE CHESS BOARD

Part of the implementation requires a means to display the chess board for a completed tour. A knight's tour is stored in a single observation that represents a two dimensional array consisting of the variables, appropriately named, (S for Solution): S11-S18 S21-S28, S31-S38…, S71-S78, S81-S88. If the tour begins in cell {3,5}, the third row and fifth column, that cell would contain the value '1'. Similarly, if the tour ended on cell {2,7}, the second row and seventh column, that cell would contain the value '64' denoting the 64th step in the tour sequence.

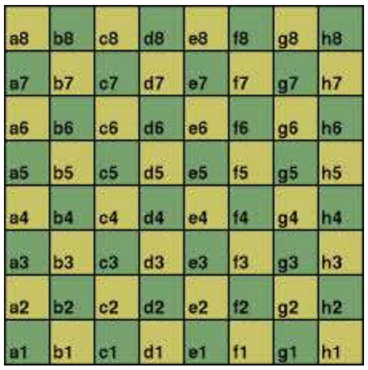
The following utility macro converts a single observation data set into a new data set containing eight observations and eight variables, denoting the rows and columns, respectively. Keep in mind that the single observation represents an abstract data type, that is, a two-dimensional array, which is used to assign the second array denoting the columns for the eight observations. The REPORT procedure generates the report, emulating a chess board.

Notice the Keyword parameters DSN and ELEMENTS. Typically, the utility is used to print the knight's tour, hence, the default values denoting the data set SOLUTION and its variables S11 through S88. However, the same macro can be used to show the possible knight moves (discussed later), using the data set KNIGHTMOVES and its variables M11 through M88.

```
%macro ShowBoard(dsn      = solution,
                 elements = s11-s18 s21-s28 s31-s38 s41-s48
                            s51-s58 s61-s68 s71-s78 s81-s88);
   data board;
      array board{8,8} &elements.;
      array cols{8} c1-c8;
      set &dsn.;
      do r = 1 to 8;
         do c = 1 to 8;
            cols{c} = board{r,c};
            end;
         output;
         end;
      keep c1-c8;
   run;
   proc report data=board nowindows headskip;
      columns c1-c8;
     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 '';
     format c1-c8 best3.1;
     title2 "&title.";
   run;
%mend ShowBoard;
```

KNIGHT MOVES

Unlike all the other pieces of chess, which move vertically, horizontally, or perhaps diagonally, the knight moves in an L-shaped fashion; that is, either two steps-turn-one step or one-step-turn-two steps. Thus, according to the depiction below, if a knight sits on square a8, it follows that the knight can land either on square b6 or c7, and nowhere else.



The proposed heuristic solution requires knowing *a priori* the number of *possible moves* by a knight at every square on the board. For example, if the knight were at square a8, there would be only two possible moves; whereas, if the knight were at square c6, there would be eight possible moves, as follows:

a8 → b6, c7

c6 → b8, d8, a7, e7, a5, e5, b4, e4

Despite the naming convention to identify a given square on the board, the SAS solution utilizes a two-dimensional array for storing the number of possible moves at each location.

But, how is the number of possible moves for a given position determined? How does the knight move without falling off the board? The Data step below accomplishes the task of creating a data set having one observation and sixty-four numeric variables M11-M18, M21-M28, …, M71-M78, M81-M88, each containing the number of possible knight moves anywhere on the board. Appropriately, there are two nested Do-loops that traverse the board, indicating the coordinate location of the knight. But then, there are two more nested Do-loops that emulate the movement of the knight. Keep in mind that a knight can move one or two steps backwards, one or two steps forward, as well as right or left. The Do-loops consider all possible combinations excluding illegal moves, that is, those instances when the absolute values of STEP1 and STEP2 (e.g. -2 and 2) are equal. Also, a Boolean expression discerns whether the knight's move lands on the board; whereupon, the move is considered viable. Ultimately, for each legal move, the Data step increments a counter variable, and ultimately stores the aggregated value in the appropriate matrix cell.

```
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;
```

The macro %ShowBoard renders a more suitable display of possible knight moves for each square. For example, the upper right corner (a8) has two possible moves, as noted earlier. Notice the symmetry of the values within the matrix, such as: the first and last rows; the second and seventh rows; and the third through sixth rows. Notice also the symmetry with respect to the upper and lower diagonal matrices. Finally, notice that there are no fewer than two and no more than eight possible moves.

Recall that the heuristic rule requires that the next viable move should be to square with minimal degree; that is, having the fewest subsequent moves. For example, square b7 (i.e. row 2 from the top and column 2 from the left) has four possible moves whose subsequent possible moves are 4 and 8. The rule would go to the square having only four possible moves. However, there are two squares that have four possible moves. The proposed solution choses *the first instance* of minimal degree – An issue that will be addressed later.

```
2    3    4    4    4    4    3    2
3    4    6    6    6    6    4    3
4    6    8    8    8    8    6    4
4    6    8    8    8    8    6    4
4    6    8    8    8    8    6    4
4    6    8    8    8    8    6    4
3    4    6    6    6    6    4    3
2    3    4    4    4    4    3    2
```

At the onset of a tour, the number of possible moves is readily apparent. However, this number changes during the tour, since a move to one square affects those squares in proximity, that is, their number of moves has decreased by one. This will be discussed in detail later, with illustration. Suffice it to say that the data set KNIGHTMOVES plays a pivotal role in finding a tour.

COMPLETING THE TOUR

The SAS solution finds a knight’s tour for every square on the chess board – Well, almost. The macro begins with two nested %DO loops creating two macro variables R and C denoting a specific starting point, such as Row 1 and Column 1. For a given starting point, the code (shown later) proceeds to find the knight’s tour and, then, displays the solution emulating the chess board.

```
%macro Warnsdorff;
   %let run = 0;
   %do r = 1 %to 8;
      %do c = 1 %to 8;

          < Find Knight’s Tour >

          < Display Sequence of Moves >

         %end;
      %end;
%mend Warnsdorff;
```

The process of finding a knight’s tour requires the KNIGHTMOVES data set and two 2-dimensional matrices MOVES, representing the number of legal knight moves for each square, and BOARD, storing the sequence number for the ith step of the tour. Notice the RETAIN statement that preserves the sequence numbers during the course of the tour.

The Data step iterates only once, since there is only one observation in the data set KNIGHTMOVES, which is used to populate the M-variables, denoting the number of moves allowed on a given square. In contrast, the S-variables have a null value, except for the initial square, located at the coordinate {r,c}, having the value ‘1’, of course. In essence, the objective is to assign the sequential values to the matrix BOARD.

```
data solution;
   retain r &r. c &c.;
   retain s11-s18 s21-s28 s31-s38 s41-s48 s51-s58 s61-s68 s71-s78 s81-s88;
   array board{8,8} s11-s18 s21-s28 s31-s38 s41-s48
                    s51-s58 s61-s68 s71-s78 s81-s88;
   array moves{8,8} m11-m18 m21-m28 m31-m38 m41-m48
                    m51-m58 m61-m68 m71-m78 m81-m88;
   set knightmoves;
   board{r,c} = 1;
```

After initialization, the process proceeds to traverse the chess board by moving to position 2 and ultimately to the last square. The loop-control variable POSITION represents the sequence number for the *next* square. But, where is the next square? Depending on the location of the knight, it can move from two to eight possible places on the board. One of those squares must be of minimal degree.

The code below begins, obviously, by moving to the second square, hopefully reaching the last square on the board. It is at this point where the code emulates the movement of a knight. Keep in mind that the knight moves either “two, one” or “one, two” making a turn along the way. Notice the nested DO-loops and the values of their loop control variables STEP1 and STEP2. Their combined values represent movements, some of which are extraneous (illegal) moves, such as: two steps, right turn, then two more steps. Nonetheless, this combinatorial approach, along with the stipulation that the absolute values of STEP1 and STEP2 are not equal, affords all the possible moves by a knight. Of course, the knight cannot fall off the board. Moreover, the square must be unused. Both conditions are accomplished by a second Boolean expression that determines a valid row, a valid column, and a viable square. Then, finally, the code proceeds to find a viable square of minimal degree.

```
     do position = 2 to 64;
        nxtr   = 0;
        nxtc   = 0;
        pmoves = 9;

        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 )            ← Valid row
                       and ( 1 le (c+step2) le 8 )        ← Valid column
                       and board(r+step1,c+step2) eq .    ← Viable square
                       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;
```

The Boolean expression below determines whether the knight landed on a viable square; that is, the variables NXTR and NXTC are valued. Otherwise, in the event of failure, the next task within the DO-block would generate the error message: "Array script out of range," since the variables NXTR and NXTC were not assigned a value between 1 and 8, accordingly. However, assuming that the knight landed safely, the code proceeds to update the matrix BOARD with the *i*th value of the variable POSITION.

```
        if 1 le nxtr le 8 and 1 le nxtc le 8
           then do;
              r = nxtr;
              c = nxtc;
              board{r,c} = position;

              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 s11-s18 s21-s28 s31-s38 s41-s48 s51-s58 s61-s68 s71-s78 s81-s88
         m11-m18 m21-m28 m31-m38 m41-m48 m51-m58 m61-m68 m71-m78 m81-m88;
  run;
```

The Data step has another very important task: Updating the MOVES matrix. Why is this important? Recall the rule: "*Always move the knight to an adjacent, unvisited square with minimal degree.*" As mentioned previously, the values denoting "minimal degree" change during the tour; and, this information must be maintained, accordingly. Consider the KNIGHTMOVES data set at the onset of a tour. As stated previously, each square represents the number of possible knight moves – From that square. So, what does it mean when a knight moves from one square to the next? Observe the three juxtaposed matrices: KNIGHTMOVES (Before move), BOARD (Move a8 to c7), and KNIGHTMOVES (After move). Focus on the numbers in red. At the onset of the tour, square a8 has three possible knight moves; however, after the move from a8 to c7, square a8 has only two possible moves, since one was used. Of course, the number of moves at a8 (2 to 1) is irrelevant, really, since it has a position value (i.e. the first move). However, it is the other unused cells that have lesser minimal degree. For example, at the onset of the tour, square a6 (third row from the top, first column from the left) had four possible knight moves, but after moving the knight from a8 to c7 (See BOARD), square c6 has only three possible knight moves, since one has been used.

MOVES (Before move)

2	3	4	4	**4**	4	3	2
3	4	6	6	6	6	4	3
4	6	8	8	**8**	8	6	4
4	**6**	8	**8**	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

BOARD (Move a8 to c7)

1	.	.	.	.	.	.	.
.	.	2	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

MOVES (After move)

1	3	4	4	**3**	4	3	2
3	4	6	6	6	6	4	3
3	6	8	8	**7**	8	6	4
4	**5**	8	**7**	8	8	6	4
4	6	8	8	8	8	6	4
4	6	8	8	8	8	6	4
3	4	6	6	6	6	4	3
2	3	4	4	4	4	3	2

Only those squares that are adjacent to the next square (i.e. Square c7) are changed, thereby affecting the next (e.g. third) move, as well as subsequent moves throughout the knight's tour. In fact, without maintaining the MOVES matrix, the implementation would fail to find any solution.

The Data Null step below discerns whether the SOLUTION data set contains a complete tour. The step creates a macro variable RESOLVED by using the N-function within a Boolean expression, assigning it either 0 or 1. Of course, a completed tour will not have any null values; whereupon, the value of the macro variable RESOLVED becomes 1, otherwise 0. Assuming a completed tour, the macro %ShowBoard renders the knight's moves on a chess board. Notice that the macro could be used unconditionally to show the outcome regardless.

```
%do r = 1 %to 8;
   %do c = 1 %to 8;
                < Aforementioned Code >
      data _null_;
         array board{8,8} s11-s18 s21-s28 s31-s38 s41-s48
                          s51-s58 s61-s68 s71-s78 s81-s88;
         set solution;
         call symput('solved' left(put(n(of board{*}) eq 64,1.)));
      run;
      %if %solved.
         %then %do;
            %ShowBoard(dsn      = solution,
                       elements = s11-s18 s21-s28 s31-s38 s41-s48
                                  s51-s58 s61-s68 s71-s78 s81-s88,
                       title    = Chess Board: Warnsdorff Rule -- Run #&run.);
         %* %ShowBoard(dsn      = solution,
                       elements = m11-m18 m21-m28 m31-m38 m41-m48
                                  m51-m58 m61-m68 m71-m78 m81-m88,
                       title    = Chess Board: Knight Moves Remaining in Run #&run.);
            %end;
      proc datasets library=work nolist;
         delete solution;
      quit;
      %end;
   %end;
```

MAKING A WRONG TURN

The proposed solution fails five times unable to complete the tour, specifically beginning from squares: g6 (See below, left), g3, f2, g2, and g1. Observe two of the failed solutions. What happened? The solution on the left indicates that the knight was unable to move after the 60th move. Why not? Obviously, because there were no valid moves available. Why not? Well, look at the square prior to the 60th move. Move number 59 had only one option, a bad one, from h5 to f4. Suffice it to say that somewhere along the way, the knight made a wrong turn. The cause for failure pertains to what are called ties of minimal degree.

Using *First* Instance of Minimal Degree

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

Using *Last* Instance of Minimal Degree

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

Recall that Warnsdorff's heuristic rule does not guarantee a solution: it's a heuristic, not a deterministic algorithm. This implementation uses the first instance of minimal degree, since the Boolean expression uses the *less than* (LT) logical operator when determining minimal degree. However, what if the last instance of minimal degree were used? Changing the LT logical operator to LE (Less than or equal) would select the last instance of minimal degree. In this case, the solution fails for those tours beginning from squares: b8, b7 (See above, right), c7, b6, b3. Fortunately, the two sets of failed runs are mutually exclusive*; therefore, the implementation succeeds in finding a solution beginning from every square on the chess board.*

CONCLUSION

Warnsdorff's rule is a heuristic method to solve the Knight's Tour problem in chess. However, this simple rule offers quite a challenge to implement, from moving the knight piece to knowing the number of viable moves for each square. The proposed SAS solution accomplishes the task, except for several cases concerning so-called ties of minimal degree. Even then, the code is capable of producing the Knight's Tour for every square, with minor tweaking.

REFERENCES

Ball, W.W. Rouse, *Mathematical Recreations and Essays*. 4th ed. Macmillan and Co., Limited.

Ganzfried, Sam, *A New Algorithm for Knight Tours*, Oregon State University.

Marateck, Samuel L., *How Good is the Warnsdorff's Knight Heuristic?,* New York University, 2008.

McKay, Brendan D., *Knight's Tours of an 8 x 8 Chessboard*, bdm@cs.anu.edu.au.

Parberry, Ian, An Efficient Algorithm for the Knight's Tour Problem, *Discrete Applied Mathematics* 73 1997 251-260.

ACKNOWLEDGMENTS

Thanks to the German mathematician H.C. Warnsdorff for his heuristic rule.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Name:	John R. Gerlach
Enterprise:	Dataceutics, Inc.
Work Phone:	609-672-5034
E-mail:	gerlachj@dataceutics.com

APPENDIX A – SEVERAL KNIGHT TOURS

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

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

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

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

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

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

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

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

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

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