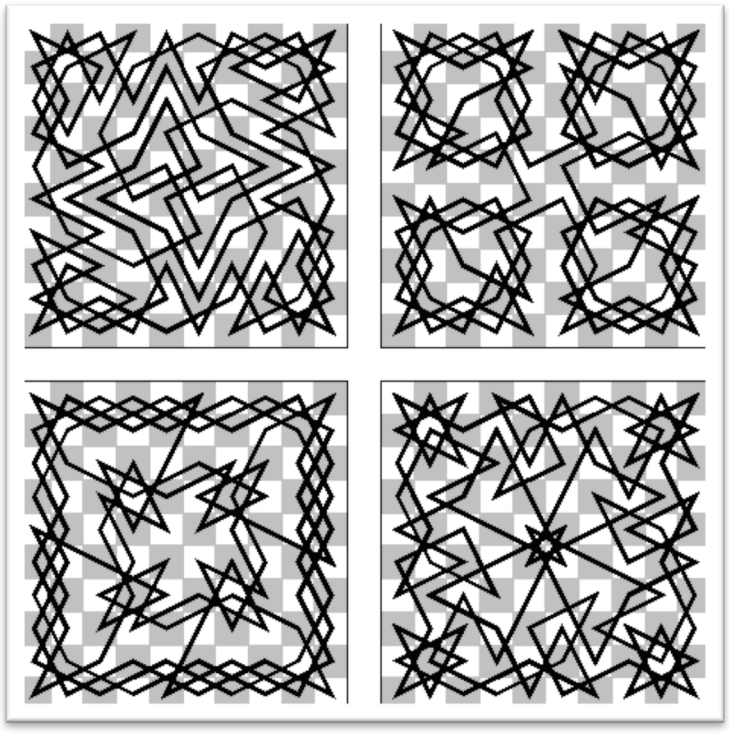
Finding and Characterizing Knights Tours on the $4 \times 4 \times 4$ Chessboard

Lydia Miller
Goshen College Mathematics Senior Seminar
Advisor: David Housman
April 2018

Introduction

The Knight's Tour Problem (KTP) is a mathematical problem that has been the subject of study for centuries. It involves a single knight moving around a chessboard. In chess, the knight can only move in an L-shaped leap that takes it 2 squares in one direction and one square in a perpendicular direction. The goal of the puzzle is to find a "tour" or path that lands a knight on every square on the board only once. A knight's circuit, or closed tour, is one in which the last move takes the knight back to the original square.

The first reference to the problem was in the 9th century AD in India, and it was revisited by Euler in the 18th century. Many further results, strategies, extensions, and proofs have been published, especially in the last few decades. Recently, mathematicians have published results related to several particularly interesting extensions of the problem that involve various types of three-dimensional chessboards.



Several knight's circuits on an 8×8 chessboard. These all have interesting symmetries, but not all knight's tours do.

Schwenk (1991) proved which $m \times n$ boards have circuits, or closed tours. Parberry (1994) published a very fast algorithm for finding knight's tours on $n \times n$ chessboards. Löbbing and Wegener (1995) actually counted the number of open and the number of closed tours on an 8×8 chessboard. DeMaio (2007) proved that closed knights tours are

present on $n \times n \times n$ cubes only when n is odd and greater than 3. Authors from Chongqing University (2009) generalize the problem much further, and prove results for the Generalized Knight's Tour, that is, an (a, b) move rather than a (1, 2) move. They do this for all dimensions $L \times M \times N$, that is, rectangular prisms. In 2010, DeMaio and Mathew completely characterized the existence of closed knights tours on $i \times j \times k$ boards.

The Problem

This paper presents algorithms for finding and characterizing knight's tours on the $4 \times 4 \times 4$ chessboard.

I'll note here that there are no tours on a $3 \times 3 \times 3$ chessboard. This is because the middle square (2, 2, 2) is totally inaccessible from any other square on the chessboard. All legal moves include a movement of 2 units in some dimension, and there is no $1 \leq a \leq 3$ for which $a + 2$ or $a - 2$ equals $1 \leq b \leq 3$.

To the right is my representation of a knight's tour on a $4 \times 4 \times 4$ chessboard. The four matrices represent the four layers of the cube. Assembled, they would stack on top of each other, with layer 1 on the bottom, and layer 4 on the top. Thus each square represents a cell in the cube, of which there are 64.

Layer 1:

6	59	26	37
57	12	55	8
48	7	36	17
11	56	63	54

To follow the tour, find the knight's starting square, indicated by the number 1. In this example, the starting square is on the fourth layer, at index (2, 4, 4). The first move takes it to the second layer, at index (2, 3, 2). The knight then continues in that manner, until arriving at the last square, labeled 64, at index (4, 4, 3).

Layer 2:

47	34	5	18
42	19	2	23
3	46	41	38
62	13	20	45

I wrote a greedy algorithm to find such tours (*Appendix 1*). I chose to use Matlab since matrices are the foundation of the language, and vectorized operations in Matlab are accessible and fast.

Layer 3:

4	25	60	27
61	58	9	16
32	35	28	53
49	10	15	64

The algorithm generates an empty $4 \times 4 \times 4$ array to represent the board, chooses a starting point randomly on the board, and chooses moves randomly from a list of legal moves. It continues in this way until a complete path is found, or until the knight is stuck (has no legal moves). At this point it starts over completely and searches for a new tour.

Layer 4:

33	30	51	24
50	43	40	1
29	52	31	22
14	21	44	39

On average, it takes around 5 seconds for the algorithm to find a tour, but this varies since the algorithm will attempt to find a tour 1,000 times, I built in a loop so that continues running 1,000 times before giving up. The algorithm fails to find a tour 0.051% of the time (sample size 1,031 runs).

Categorization

Due to the symmetry of a cube, there are many tours that could be equivalent, but would look different when represented as above. For my purposes here, I consider tours to be equivalent if they have the same "tour type" as described below. However, it might be interesting in a further study to define equivalent tours as those for which a linear transformation exists mapping one onto the other.

In order to categorize tours, I made a list of all the 3 dimensional indices in the cube listed in the order in which they were visited. Then I replaced it with a code: a number from 0 to 3 representing its location on the cube. The code for this process is in *Appendix 1*. The location codes relate to the number of exterior faces for that location, as shown in the table below.

Location Code	Number of cells	Description
3	8	3 exterior faces – these are the corners of the cube.
2	14	2 exterior faces – these lie along the edges of the cube
1	24	1 exterior face – these lie in the middle of the outside faces
0	8	No exterior faces – these are completely enclosed in the middle of the cube.

The tour types are thus ordered lists of 64 location codes. Two tours might look different in their original form, but when represented as a list of location codes, they are exactly the same. These would be tours with the same tour type, that is, equivalent tours by my definition.

There are a few examples of tours whose location code lists might look different, but which are equivalent tours.

1. Tours in backwards order: I consider tours to be equivalent if the tour type of one is equal to the reverse of the tour type of the other. Matlab's command `fliplr()`, flips a vector left to right, which is how I checked for this type of equivalency.

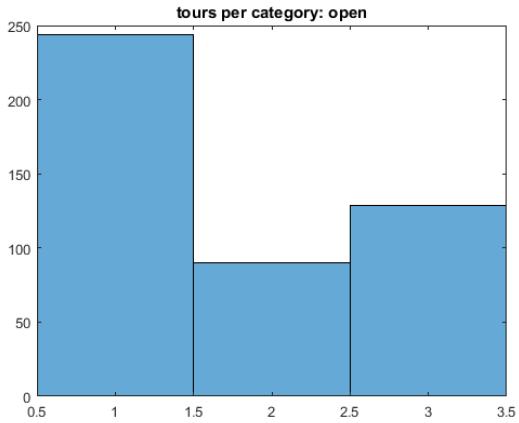
2. Cycles: Also known as closed tours, cycles are those where the tour ends where it starts, so there could be many tours that are part of the same cycle, and thus equivalent, but start and end at different places. To check for this type of equivalency, I used Matlab's `circshift()` command.

Results

I found and categorized 1,000 tours, and the results were as follow:

Open Tours consisted of 831 out of 1,000

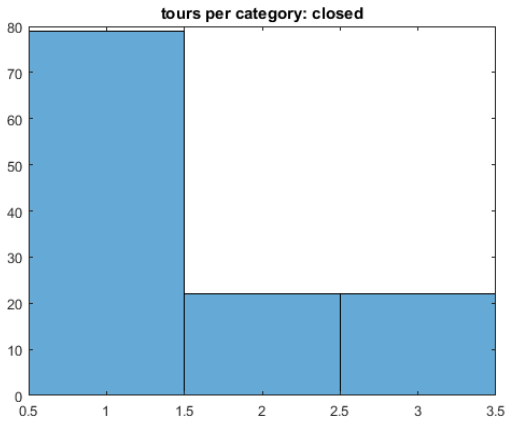
There were 463 categories of open tours



About half the tour types contained only one tour of that type, and about half contained two or three.

Closed Tours consisted of 123 out of 1,000

There were 123 categories of closed tours



Most of the tour types contained only one tour of that type, and about a third contained two or three.

Conclusion

This paper presents an algorithm for finding knight's tours on a $4 \times 4 \times 4$ cube, using a very simple greedy algorithm. It also presents a definition and method for categorizing tours. Two further areas of study would be particularly good extensions of this work. First, improving the tour-finding algorithm to be more systematic and fast. And second, defining equivalence in terms of linear transformations, and categorizing tours in this way. As always, there are many more extensions to be found by varying the dimensions, so going to larger cubes or higher dimensions could prove interesting as well.

Works Cited

Shwenk, A. J. "Which rectangular chessboards have a knight's tour?" *Mathematics Magazine,* vol. 64, no. 5, pp. 325-332, 1991.

J. DeMaio and B. Mathew, "Which chessboards have a closed knight's tour within the rectangular prism?" *Electronic Journal of Combinatorics*, vol. 18, no. 1, pp. 8-14, 2011

S. Bai, X.-F. Yang, G.-B. Zhu, D.-L. Jiang, and J. Huang, "Generalized knight's tour on 3D chessboards," *Discrete Applied Mathematics,* vol. 158, no. 16, pp. 1727-1731, 2010

I. Parberry. "An efficient algorithm for the Knight's tour problem," *Discrete Applied Mathematics*. Vol 73, no. 2, pp. 251-260. 1997

M. Löbbing and I. Wegener, "The Number of Knight's Torus Equals 33,439,123,484,294 – Counting with Binary Decision Diagrams," *The Electronic Journal of Combinatorics,* vol. 3, no. 1, 1996.

Appendix 1: Code

Main Tour-Finding Algorithm:

```
for m = 1:1000
    i = 2;
    cur_pos = randi([1,4],1,3);
    board = zeros(4,4,4);
    board(cur_pos(1),cur_pos(2),cur_pos(3)) = 1;
    success = true;
    found_tour = false;
    for n = 1:65
        if success
            ml = GenerateMoves(cur_pos);
            while ~isempty(ml)
                choice = randi([1,size(ml,1)]);
                move = ml(choice,:);
                if board(move(1) ,move(2), move(3)) == 0
                    new_pos = move;
                    board(new_pos(1),new_pos(2),new_pos(3))
                    = i;
                    cur_pos = new_pos;
                    i = i + 1;
                    success = true;
                    break
                else
                    ml(choice,:) = [];
                    success = false;
```

```
                end
            end
        else
            break
        end
    end
    if i-1 == 64
        found_tour = true;
        break
    end
end
if found_tour
    tours = cat(4,tours, board);
end
stats = [stats;[m, n, found_tour]];
```

GenerateMoves (supporting function for the main algorithm)

Given a current position, generates all the possible legal moves that would keep the knight on the board and puts them in a list called ml (moves list)

```
function ml = GenerateMoves(cur_pos
     p = [perms([0 1 2]);perms([0 -1 -2]);perms([0 -1
         2]);perms([0 1 -2])];
    moves = p + cur_pos;
    ml = [];
    for i = 1:length(moves)
     if isequal((moves(i,:) > 0)+(moves(i,:) < 5),[2,2,2])
            ml = [ml;moves(i,:)];
        end
    end
end
```

find_index (given a value, gives the 3 dimensional index location)

```
function ind = find_index(board, val)
    [a,bc] = find(board == val);
    b = rem(bc,4);
    if b == 0
        b = 4;
    end
    c = floor(bc/4)+1;
    if c == 0
        c = 4;
    end
    if c == 5
        c = 1;
    end
```

```
    ind = [a,b,c];
end
```

Categorization Process:

```
open = [];
closed = [];
list = tours;
for i = 1:size(list,4)
    cycle = false;
    match = false;
    board = list(:,:,:,i);
    tt = tour_type(board);
     if LegalMoveQ(find_index(64,board),
     find_index(1,board))    % if it's a cycle
        for j = 1:size(closed,1)
            for k = 1:64
                if isequal(closed(j,2:end),circshift(tt,k))
                    % matches with another category
                    closed(j,1) = closed(j,1) + 1;
                    match = true;
                    break
                end
            end
        end
        if ~match % new category
            closed = [closed;1,tt];
        end
    else
        for j = 1:size(open,1)
          if isequal(open(j,2:end),tt) ||
          isequal(open(j,2:end),fliplr(tt))
                % matches with another category
                open(j,1) = open(j,1) + 1;
                match = true;
                break
            end
        end
        if ~match %  new category,
            open = [open;1,tt];
        end
    end
end
```

TourType function

```
function outlist = tour_type(inmat)
    outlist = [];
```

```
    for i = 1:64
        ind = find_index(i,inmat);
        category = sum(ind == 1) + sum(ind == 4);
        outlist = [outlist,category];
    end
end
```

LegalMoveQ (determines whether a move is a legal knight's move)

```
function lm = LegalMoveQ(cur_pos,new_pos)
    d = sort(abs(new_pos-cur_pos));
    lm = sum(d == [0,1,2]) == 3;
end
```