TERM PROJECT

ADVANCED GRAPH THEORY

KNIGHTS TOUR

Chaitanya Nitin Bhutada [17CS10011]
Sanket Rajendra Meshram [17CS30030]

Guided By - Prof. Bhargab B. Bhattacharya

Computer Science and Engineering
Indian Institute of Technology , Kharagpur

20/11/2020

ABSTRACT

The "Knight's Tour" problem, to move a knight on a chessboard or a n*m board so as to cover all the squares exactly once.

INTRODUCTION

We will be discussing 3 Algorithms and Comparing their Space and Time Complexities.

1] Backtracking:

Marking the Current state visited go the next non-visited state and if we can complete the tour by this visit we return the Order of visiting or else we go to the next non-visited state.

2] Warnsdorff's Rule:

Taking the Backtracking Algorithm as our base we apply a heuristic of accessibility to get our solution faster.

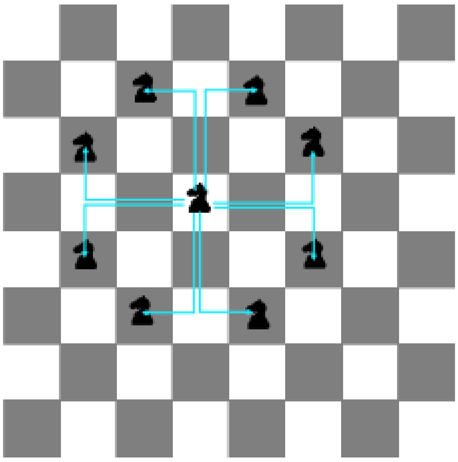
3] Neural Network:

With the help of Neurons we will configure the network so as to restrict degree of each node to be 2. The state when we reach this is called a Stable state. The neurons are updated sequentially by counting squares on the chess board in row-major order and enumerating the neurons that represent knight moves out of each square.

All the implementations can be found : here

BACKTRACKING

Starting from a random square. We mark this square as visited and move until we have covered all the n*m positions.



As in the diagram we can see the knight has at-most 8 next moves. For the current position we mark it as visited if we have covered all the n*m squares the sequence is returned else we move to the next non-visited position. If this move does not fetch us all the n*m squares we move to the next non-visited position.

The pseudo code is as follows:

```
def solveKnightMove(board, n, move_no, currRow, currCol):
        board[currRow][currCol] = move_no
        if move_no == (n*n):
                return True
        for i in range(8):
                nextRow = currRow + rowDir[i]
                nextCol = currCol + colDir[i]
                if canPlaceKnight(board, nextRow, nextCol, n):
                        board[nextRow][nextCol] = move_no + 1
                        isSuccessfull = solveKnightMove(
                            board, n, move_no+1, nextRow, nextCol)
                        if isSuccessfull:
                                return True
                        board[nextRow][nextCol] = 0
        return False
```

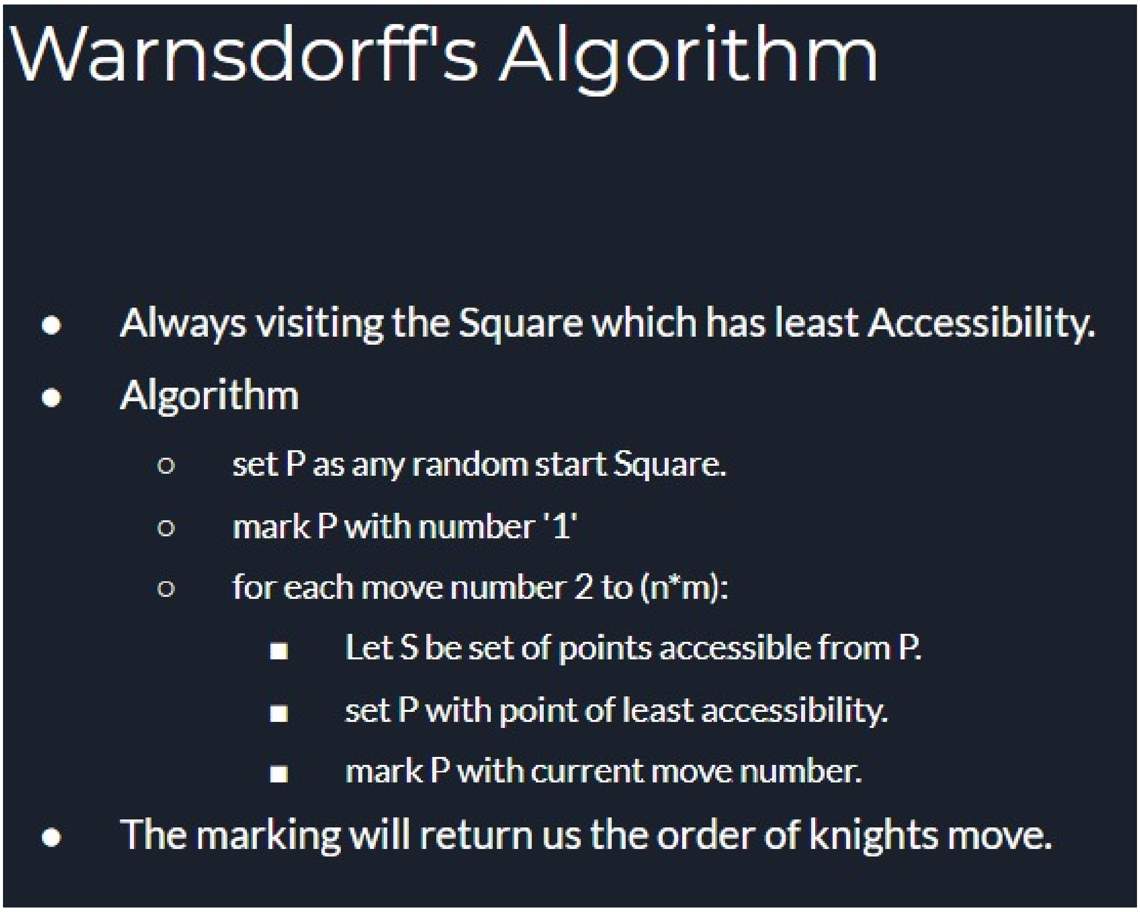
Complexity Analysis

The space complexity is linear i.e O(n*m) as we just require to store the visited sequence. Time complexity is $O(8^{(n*m)})$ Time taken for our implementation for 6*6 was 540 secs.

WARNSDORFF'S RULE

- Number of Directed knight's tour increases rapidly .
- Warnsdorff's rule uses this fact.

n	Number of directed tours (open and closed) on an *n* × *n* board (sequence A165134 in the OEIS)
1	1
2	0
3	0
4	0
5	1,728
6	6,637,920
7	165,575,218,320
8	19,591,828,170,979,904



- This gives the solution for 100x100 within 1 min .

Neural Network

The neural network is designed such that each legal knight's move on the chessboard is represented by a neuron. Therefore, the network basically takes the shape of the knight's graph over an n×n chess board. (A knight's graph is simply the set of all knight moves on the board)

Each neuron can be either "active" or "inactive" (output of 1 or 0). If a neuron is active, it is considered part of the solution to the knight's tour. Once the network is started, each active neuron is configured so that it reaches a "stable" state if and only if it has exactly two neighboring neurons that are also active (otherwise, the state of the neuron changes). When the entire network is stable, a solution is obtained. The complete transition rules are as follows:

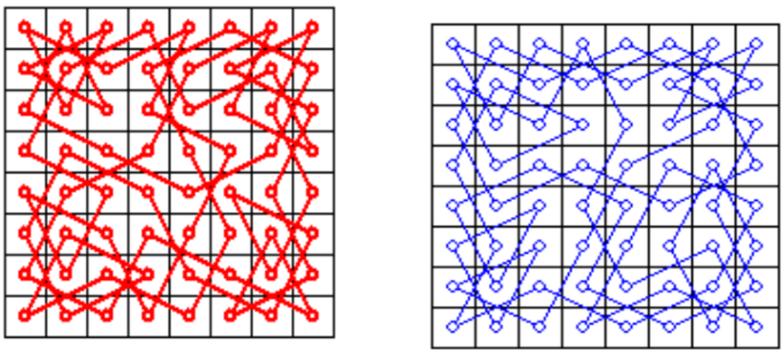
$$U_{t+1}(N_{i,j}) = U_{t+1}(N_{i,j}) + 2 - \sum_{N \in G(N_{i,j})} V_t(N)$$

$$V_{t+1}(N_{i,j}) = \begin{cases} 1 & U_{t+1}(N_{i,j}) > 3 \\ 0 & U_{t+1}(N_{i,j}) < 0 \\ V_{t+1}(N_{i,j}) & otherwise \end{cases}$$

where t represents time (incrementing in discrete intervals), $U(N_{i,j})$ is the state of the neuron connecting square i to square j, $V(N_{i,j})$ is the output of the neuron from i to j, and $G(N_{i,j})$ is the set of "neighbors" of the neuron (all neurons that share a vertex with $N_{i,j}$).

Initially (at t=0), the state of each neuron is set to 0, and the output of each neuron is set randomly to either 0 or 1. The neurons are then updated sequentially till we get a stable solution.

The network is configured to give subgraphs of degree 2 within the Knight Graph. The set of degree 2 subgraphs include closed knight tour. However there are many other solution that are not knights tour. For example there may be two small independent circuit in knights graph.



(a) Knights tour **(b)** 4 independent circuits

Figure 1: 2 possibilities

Complexity Analysis

The space complexity is linear O(n*m) i.e size of the board as the Knights graph is O(n*m). For $n \leq 20$ the number of iterations needed to get a stable network is < 100. For n = 26 the

probability of getting a knights tour as a subgraph is 1 out of 40000 so as the n increases using this approach would not be feasible as number of non-solutions would increase.
For our implementation time taken for 8*8 was = 9.725 sec.

General Knight Tour

1. Boards of Width 3

Theorem : There does not exists a tour on a 3 × 3 , 3 x 5 , 3 x 6 board .
Proof : Run BACKTRACKING on these boards .
Theorem : There exists a tour on a 3 × m board unless m = 3, 5, 6.
Proof : We will show a tour beginning in the upper left for boards of size m = 4, 7, 9, 10. These boards can then be connected together to form all possible 3×m boards except m = 3, 5, 6, all of which were proven to be impossible.

1	4	7	10
8	11	2	5
3	6	9	12

The 3 × 4 board

1	14	17	20	11	8	5
16	19	12	3	6	21	10
13	2	15	18	9	4	7

The 3 × 7 board

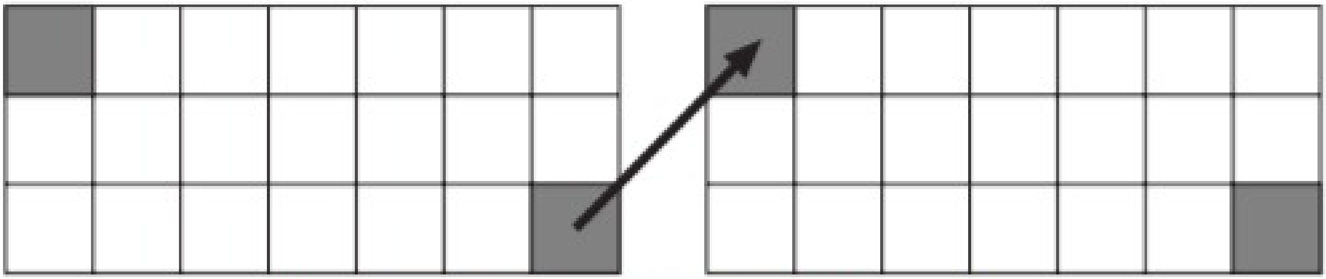
1	14	17	10	7	4	19	22	25
16	9	12	3	18	23	26	5	20
13	2	15	8	11	6	21	24	27

The 3 × 9 board

1	4	7	22	15	20	13	26	29	18
8	23	2	5	10	25	16	19	12	27
3	6	9	24	21	14	11	28	17	30

The 3 × 10 board

Given any 3×m board where m 6= 3, 5, 6, the above boards can be strung together where the numbers on the boards are incremented appropriately.



Two 3 × 7 boards hooked together

Claim : All 4xm Tour on board can be represented as concatenation of 3 × 3 , 3 x 5 , 3 x 6 board Tours.

Proof :

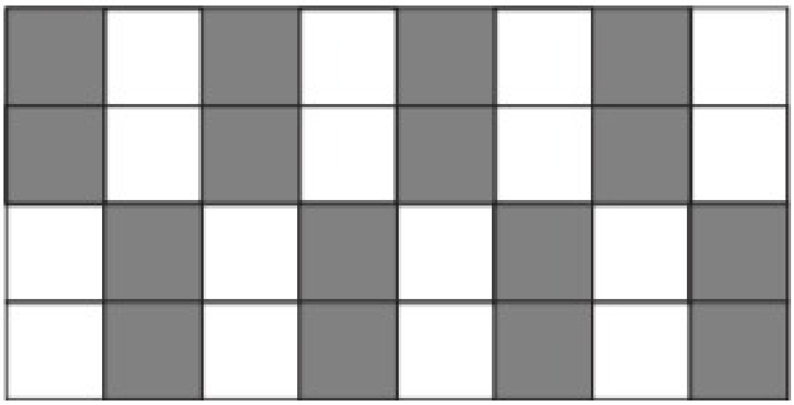
- 11 = 7+ 4
- 12 = 4 + 4 +4
- 13 = 9 + 4
- 14 = 10 + 4
- 15 = 4 + 4 + 7
- 16 = 4 + 4 + 4 + 4
- 17 = 10 + 7
- 18 = 9 + 9
- 19 = 7 + 4 + 4 + 4
- 10(n+1) + k = (n*10) + (10 + k) ; 10+k can be replaced from above .

Hence Proved

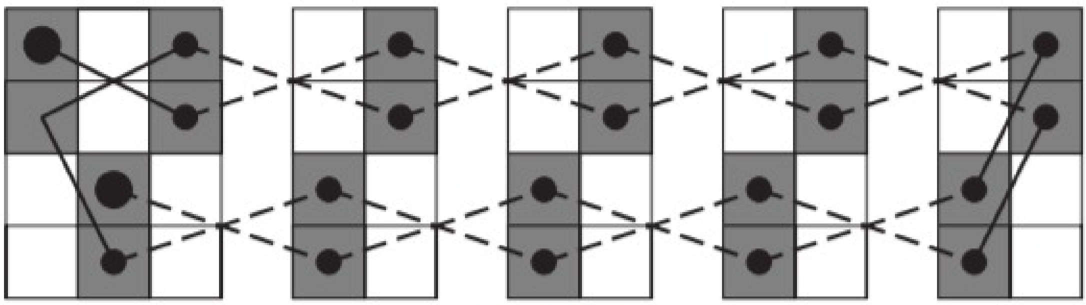
2. Boards of Width 4

Theorem : The grey squares can be toured on a 4 × m board for m 5.

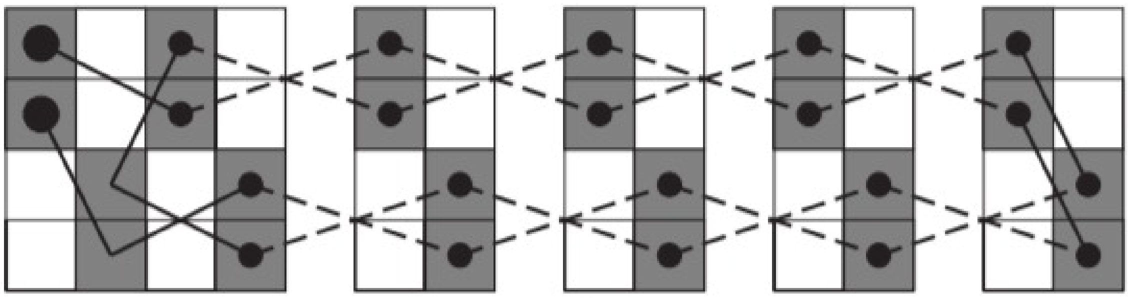
- Divide the board into colour white and grey .
- Make first two in first column grey and next 2 as white or vise versa .
- Then keep alternating on next columns.



- Tour on grey square .

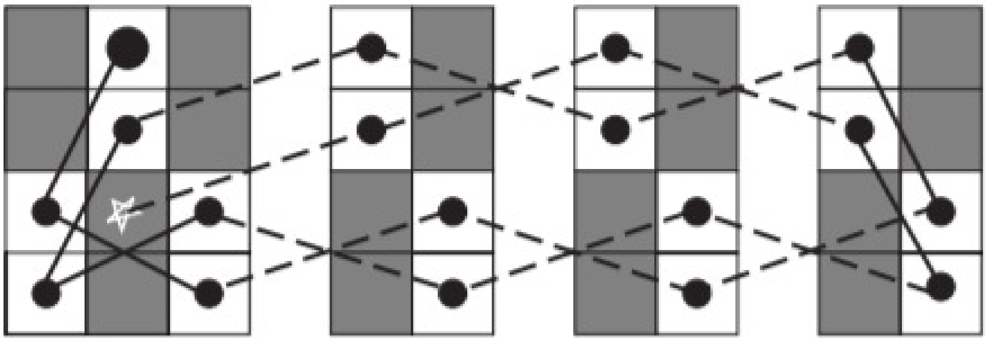


Odd Case

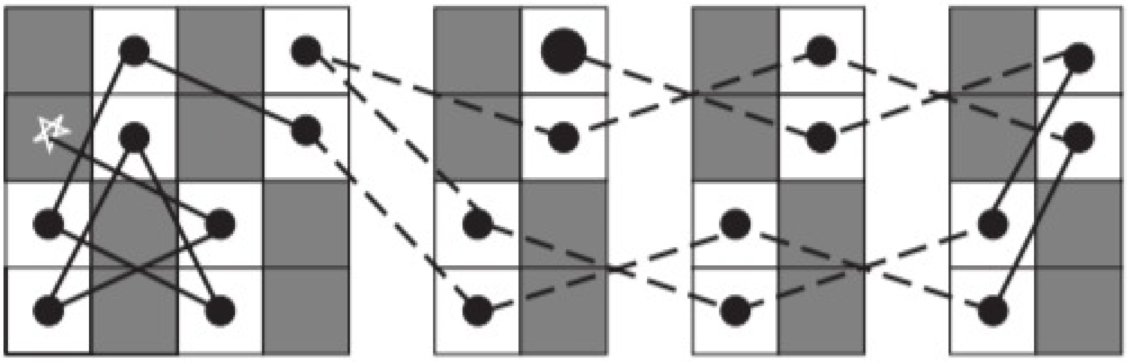


Even Case

- Tour on white squares.



Odd case



Even case

- Star represent square where Tour on Grey squares end .

- First travel all the grey squares as shown above . Then all the white squares . Which completes the Tour .

REFERENCES

1. SAM GANZFRIED "A SIMPLE ALGORITHM FOR KNIGHT'S TOURS"
3. Knight's Tour by Kevin McGown
3. Number of knight tour's
4. Neural network computing for knight's tour problems