CMSC 142 MP2: Knight's Tour

Juancho Meneses
Gimel Velasco
jrv.meneses@gmail.com
gfvelasco@up.edu.ph

ABSTRACT

This paper presents an application of a blind search algorithm to the Knight's Tour. For analyzing the performance of the algorithm, the behaviour of the time it takes to complete a tour starting from 5 different positions is observed.

1 INTRODUCTION

Considering its flexibility, the knight is the most special piece in chess. It is the only piece that can jump over other pieces and its movement is not in a straight line which allows it to attack a queen, bishop, or rook without being attacked by that piece in return. Its move is shaped as an fiLfi (two squares horizontally or vertically and then one more square perpendicular to the previous direction).[1] A knight's tour is a sequence of 64 knight moves that visits each square exactly once.[2]

The objective of this paper is to implement a blind search algorithm of the Knight's tour and analyze the results by recording the time for a solution to be found on different starting positions. The methods used in developing and writing the program and the results obtained from the program will be discussed in the next sections.

2 METHODOLOGY

There are many kinds of Blind Search Algorithms or Uninformed Algorithms. Such algorithms are the Depth-First Search (DFS) Algorithm, Breadth-First Search (BFS) Algorithm, Uniform Cost Search (UC), Depth-Limited Search (DL), Iterative Deepening (ID) Search and Bi-directional (BD) Search. For the Knight's Tour Puzzle/Problem, the Depth-First Search is implemented since it is the kind of blind search algorithm that best fits to be able to solve the Knight's Tour Puzzle efficiently even though all of the Uninformed Algorithms will be able to solve the Knight's Tour problem. Compared to the breadth-first approach, the Depth-First approach will be faster in getting a solution since it will go straight into checking a path rather than going through all of the other possible paths. This reason also applies to the Depth-Limited Search and Iterative Deepening Search since there have the same approach like that of a breadth-first search. Now for the Bi-directional Search, the program would be much more complicated to do since it needs to consider the end point of the path–which is not known. But for solving a closed toured Knight's Tour problem, it would be much more useful since the 64th step of the Knight could be easier to predict. So with all the reasons stated above, the Depth-First Search Algorithm is implemented in the problem so that the Knight's Tour puzzle could be solved.

The pseudo code of the program is shown below.

Algorithm 1 Pseudo Code of Depth First Backtracking

```
Put Knight in first tile
while Knight doesn't have 64 steps do
    Do until there is no more tile to step on
    Jump to the next unvisited tile
    Go back and try the next neighbor
end while
```

The program was created using C programming language. A snippet of the code of the program is shown in Figure 1. The rest of the code is also sent together with this paper.

```
int pathfinder_dfs(struct node **h, struct node *kpos, int knight_jump, int tree_step){
    int i,j,nctr,result;
    ////////////////////////////////////////////////
    struct node *p = kpos;
    p->visited = knight_jump;
    if((tree_step-1)%500 == 0){
        system("cls");
        print_board(h);
    }
    if(knight_jump == 64){
        p->visited = 64;
        return 1;
    }
    for(nctr=0;nctr<8;nctr++){
        if(p->neighbors[nctr] != NULL){
            if(p->neighbors[nctr]->visited == 0){//if neighbor is unvisited, move there
                struct node *t = p->neighbors[nctr];
                if(pathfinder_dfs(h,t,knight_jump+1,tree_step+1) == 1){
                    return 1;
                }
            }
        }
    }
    p->visited = 0;
    return 2;
}
```

Figure 1: Code Snippet

The data structure is given in Figure 2.

```
struct node{
    int row,col; //row and column
    int visited; //0 - unvisited | 1 - visited
    struct node *neighbors[8]; //the 8 possible tiles the knight can jump to
    struct node *templink; //a linear link just for conveniece of accessing all the tiles
};
```

Figure 2: Data Structure

The program was run in a Toshiba laptop with Intel(R) Core 2 Duo CPU(2 core(s)) processor, with 4 GB RAM and 64bit operating system. The program asks for the user to input the starting point of the Knight in the 8x8 board. The program was run three times in each of the following inputs: A1, A8, H1, H8, and D4. Other applications such as Google Chrome, Windows Explorer, Spotify, and Sublime Text were also running while the program was being ran.

3 RESULTS AND DISCUSSIONS

The program outputted the solved board and the completion time of the solution. The results of the three runs of the different starting positions are can be seen in the following figures. The running times of the runs of the starting positions is shown in the table below.

Starting Position	Runtime (ms)
A1	20484
	20597
	20514
H1	2251
	2257
	2261
A8	285
	287
	286
H8	741
	726
	722
D4	15899629
	18502156
	17191270

Figure 3: Path from A1

Since the program did not randomize in choosing a neighbor position, the path generated by the three runs of a starting position input were the same. As can be observed in the table, starting at the four corners took only few seconds for the knight to complete the tour. It is reasonable because there are only two neighbor moves at the corners which means there are only two traversals needed to find a complete 64 jumped positions without repetitions.

3.1 Case of Starting Position D4

As can be seen in the previous table, starting at the position D4 took a very long time for the knight to complete the tour compared to previous runs starting at the corners. This is because there are eight neighbor moves in that starting position which also means that it must perform eight traversals of the neighbor moves to find a path which completes the tour. (See similar figure in figure 8)

Figure 4: Path from A8

Figure 5: Path from H1

Figure 6: Path from H8

The eight traversals mean that the knight will go to one of the eight neighbor moves at the start and then continuously go to other unvisited neighbor moves until the tour is completed but if starting at that neighbor move did not give a solution, the knight will go back to the starting position which in our case is D4 and it will go to an unvisited neighbor move of the seven remaining neighbor

```
Path of Knight's Tour starting in D4:
R C|  A   B   C   D   E   F   G   H
 8 |  52  57  48  39  50  59  34  41
 7 |  47  38  51  58  35  40  25  60
 6 |  56  53  36  49  26  33  42  23
 5 |  37  46  27  54  43  24  61  32
 4 |  28  55  44  01  62  31  22  07
 3 |  45  02  11  30  19  08  63  16
 2 |  12  29  04  09  14  17  06  21
 1 |  03  10  13  18  05  20  15  64
```

Figure 7: Path from D4

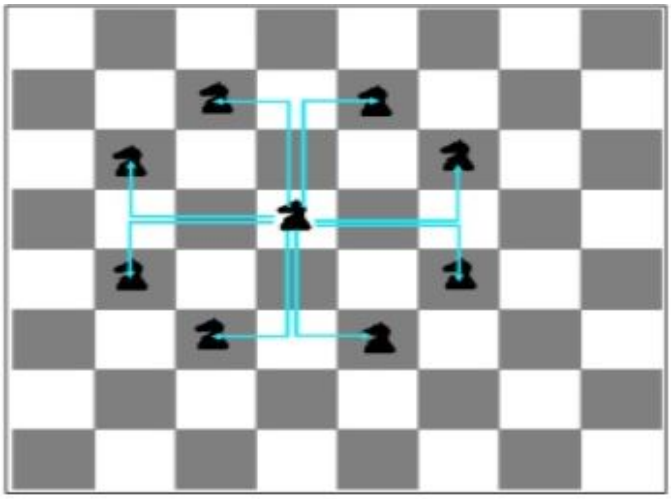

Figure 8: Similar case of D4 start

moves at the start and it will go on and on until it finds a solution which completes the tour.

4 CONCLUSIONS

Completing the Knight's tour starting from the four corners of an 8x8 board took only few seconds using a blind search algorithm. Starting at D4 or somewhere at the middle will take long hours to complete the Knight's tour with the use of a blind search algorithm. The reliability of the algorithm depends on where the starting position is in terms of time complexity.

REFERENCES

[1] Piece Movement. https://docs.kde.org/trunk5/en/extragear-games/knights/piece-movement.html. (????). Accessed: 2017-04-06.

[2] Adam Berliner. 2011. A Knight's Tour de Force. *Math Horizons* 18, 4 (April 2011), 27–29. DOI: http://dx.doi.org/10.4169/194762111x12986558508975