# *A Simple Recursive Backtracking Algorithm for Knight's Tours Puzzle on Standard 8×8 Chessboard*

Debajyoti Ghosh
Computer Science
NIIT University
Neemrana, India
4u.debajyoti@gmail.com

Uddalak Bhaduri
Computer Science
NIIT University
Neemrana, India
bhaduriu@gmail.com

***Abstract*—In this paper, we propose a simple recursive backtracking algorithm and compare its performance against existing algorithms for a classical chessboard puzzle in recreational mathematics known as knight's tour on standard or regular 8×8 square chessboard. In this work, we re-investigate this celebrated problem with the help of modern computer. Our algorithm is easy to implement compared to other existing algorithms for knight's tour puzzle. It has previously been proven that a knight's tour is always possible for sufficiently large sized chessboards either rectangular or square, but the novelty of our work lies in its simplicity and the fact that here we restrict knight's tour for 8×8 chessboard only.**

***Keywords—Algorithms, Backtracking, Recursion, Heuristics, Knight's tour, Chess puzzle***

## I. INTRODUCTION

In the 18th century, the outstanding Swiss mathematician, Leonhard Euler before the advent of computers first introduced and investigated this puzzle and gave a mathematical analysis. His approach was to develop tours by deriving new tours from existing tours.

On recreational mathematics, mathematical puzzles play a central part and drift many of us towards solving challenging puzzles. One of the most interesting classic chessboard puzzle for chess buffs is the knight's tour puzzle. The classic combinatorial optimization problem, knight's tour is a classical ancient puzzle concerning chessboards which has been studied for centuries. Starting from Euler, several other researchers, like De Moivre, Vandermonde, Warnsdorff, Roget and many more also investigated knight's tour in recent years.

The knight's tour puzzle has fascinated chess players, mathematicians and computer scientists alike for many years. Chess is a two-player game played on a square board of size 8×8. The standard chessboard has 64 squares arranged in 8 equal rows and columns, and have alternating light and dark colors. Alternatively, we can say that a regular chess is played on a square board divided into an 8×8 grid. For knight's tour problem, the individual square colors are immaterial and the entire board is just treated as a uniform 8×8 grid. In a regular chess game, there are six types of pieces, namely the King, Queen, Knight, Bishop, Rook, and Pawn. Each one of these pieces has its own unique rule to move that is predefined.

In the game of chess, the 'knight' (♘ ♞, generally referred to as a horse due to this resemblance) is the most special piece that represents a knight and often depicted as a horse's neck and head. The knight moves in an arbitrary oriented L-shaped pattern direction, jump over all other pieces in its way to reach a square on a chessboard which is empty. A knight always moves by skipping two squares horizontally and then one square vertically or two squares vertically and then one square horizontally to whichever direction (positively or negatively) it moved two positions, i.e., it can move either one column over and two rows over, or one row over and two columns over in either direction but in either case, it cannot cross boundary, i.e., the edge and corner of the board. In a standard 8×8 empty chessboard, but for a single knight on some square. The number of possible knight moves from a given square, where it currently resides, is at most eight, which are adjacent to the current square, depending on the location of the square within the chessboard. More, specifically, if the knight is on square (a, b), it is allowed to move on one of the eight possible squares $(a \pm 1, b \pm 2)$ and $(a \pm 2, b \pm 1)$, provided all these squares are within the board. Formally, (a, b) is adjacent to (c, d) if and only if $0 \le a,b,c,d \le n-1$, a and b are index of the current square and $(c, d) \in \{(a-2, b+1), (a-1, b+2), (a+1, b+2), (a+2, b+1), (a+2, b-1), (a+1, b-2), (a-1, b-2), (a-2, b-1)\}$. From these eight moves, we observe that the knight's move is symmetric, i.e., if a knight can move from a square index (i, j) to a square (i', j') in one move, it can also move from (i', j') to (i, j) in one step. Our task is to emit a series of admissible knight moves that result in the knight visiting every square on the chessboard only once. The eight possible types of move for a knight from a given initial position are shown in the following diagram, where ♘ denotes the given initial position of the knight:

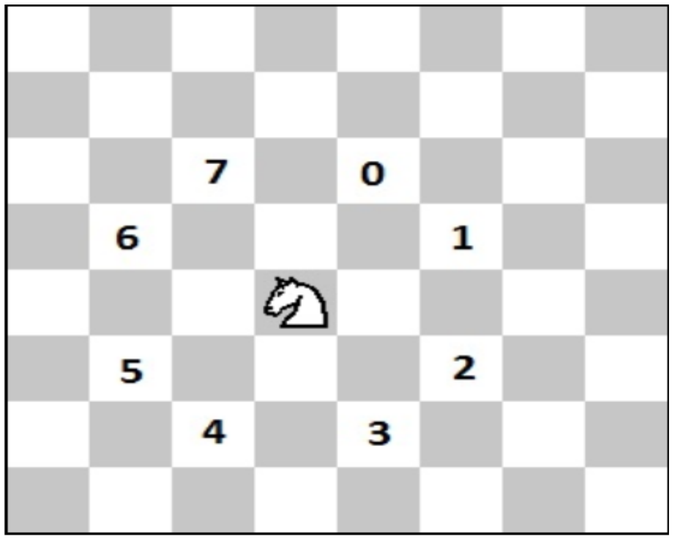

Fig. 1: The 8 Possible moves patterns for knight on an 8×8 chessboard.

The objective of the legendary knight's tour puzzle is to construct a sequence of admissible moves of a lonely knight starting from an arbitrary square of an entire 8×8 chessboard such that it visits all the other 63 squares in exactly 63 moves starting from any arbitrary square on the board and visiting on the remaining squares of a chessboard only once such that every square appears in the derived sequence exactly once. The output should indicate the order in which the knight visit the squares, starting with the arbitrary given initial position. Note that it is not required that the tour be "closed" always.
Applications of the knight's tour includes generating pseudo random numbers, multimedia image encryption and compression, secure data hiding, steganography, signal processing, digital image information security, etc.

## II. Related problem in Graph Theory

When the knight's tour is represented as a graph, it is simply an instance of Hamiltonian path in an arbitrary undirected graph, where the task is to find whether there exists a path such that every vertex is visited exactly once. The squares of the board represent the vertices. The knight's permissible moves between adjacent squares are represented by edges. In analogy, knight's tour puzzle is to find a path on the chess board where the knight traverses each square only once. It is equivalent to finding a Hamilton path in knight's graph. Whenever corresponding knight's graph has a Hamiltonian path, then a knight's tour exists on the chessboard. It is suspected till date that finding Hamiltonian path is computationally hard problem, i.e., NP-complete problem [7], that's why no known efficient (polynomial time) algorithm exists so far. It is shown that this problem is NP-complete by reducing from the Hamiltonian path problem. Unlike the general Hamiltonian path problem, the Knight's tour problem can be solved in polynomial time [1]. Open knight's tour is referred to as Hamiltonian path and closed knight's tour is referred to as Hamiltonian cycle. Similarly, finding a closed knight's tour is an instance of the Hamiltonian cycle finding problem.

## III. Number of tours

The 'knight' travels in the chessboard in such a way that it visits successively on to all possible square exactly once, and complete a knight's tour [30]. Writing a program to find a knight's tour is one of the common programming problems given to computer science students. There are two kinds of knight's tour, a closed knight's tour and an open knight's tour, defined as follows: A knight's tour is called re-entrant or closed if the knight can reach last traversed square from the starting square by only one knight's move, otherwise it is called an open tour. The knight's tour has an amazingly large number of solutions. According to Euler, "Although the number of these routes is not infinite, it will always be so great that one could never exhaust it". One can also show that no closed knight's tours exist for boards of some dimensions like 3 × 6, 3 × 8, and 4 ×n, etc. Similarly, no open knight's tours exist for boards of some dimensions like 3 × 5, 3 × 6, 3 × 3, and 4 × 4, etc. A (open or closed) knight's tour is not possible on a 4×4 chess board. To determine the number of solutions for the knight's tour on a standard chessboard of size 8 × 8, the number of directed closed tours are exactly 26,534,728,821,064 (as are rotations and reflections, opposite directions are counted separately). Obviously, the number of undirected closed tours is half this number, i.e., 13,267,364,410,532, that is approximately $10^{13}$ since each tour can be traced in reverse, calculated in 2000 by Wegener [22]. Just three years before this, in 1997, the same number was claimed by Brendan McKay [20]. Obviously, any square on the given chessboard can be taken as the starting point whenever a knight's closed tour exists. The number of open tours on a standard 8 × 8 chessboard is unknown so far, as a result remaining an open question. A brute-force search for a knight's tour is unrealistic apart from the smallest boards; for example, on an 8×8 board there are nearly $4\times10^{51}$(392635605334300583964134272930853505712708 38 75101072, to be exact) move sequences possible, which is far beyond the computing capacity of modern digital computers to execute operations on such a huge set. The task is to devise an efficient algorithm that produces all the admissible knight's tour starting from any arbitrary position. Knuth gave a solution for the longest uncrossed knight's tour over an 8×8 chessboard [14], which was posed by T. R. Dawson once, but L. D. Yardbrough launched the same problem again in the July 1968 in the Journal of Recreational Mathematics.

## IV. Past work

Several researchers tried to find efficient algorithms to resolve the knight's tour puzzle as well as to count the number of tours with computers. Some algorithmic and heuristic solution already exists to find knight's tour on a given board with the help of a computer. In 1759, Euler first presented a systematic solution for the 8×8 board [27]. In October 1967, in column of Scientific American [28] Martin Gardner wrote about the knight's tour on rectangular boards and knight related other mathematical problems. German mathematician H. C. Warnsdorff [5] in1823 proposed a simple greedy heuristic to determine the next unvisited square to be selected is the one that has the fewest number of available moves rather than using random selection, does not require backtracking and finds one solution immediately. Warnsdorff's rule suggests – "Always move to an adjacent, unvisited square with minimal degree". Naive backtracking is generally slow, as it can simply reach dead-end and has to re-evaluate many decisions. Optimization of the naive algorithm is possible by applying the Warnsdorff's rule. Warnsdorff's rule for a knight's tour does not produce the desired result all the time, but fails for n>49 [6]. Flaw of Warnsdorff's rule for finding knight's tours on chessboard is that, it does not provide a tiebreaking rule, even the rule was not a deterministic one. In 1967, Pohl [2] proposed an improvement over the Warnsdorff's rule, and proposed a selection rule in case of ties, a tie-breaking method. In 1994, Conrad et al. [3] proposed a much faster but complicated algorithm and finds the closed knight's tour on the n × n boards for n ≥ 5. In 1997, Parberry [1] gave a polynomial time, simple divide-and-conquer approach for knight's tours of various types (for all n ≥ 6). The modern approach to solve the knight's tour problem is to use a genetic algorithm. In 2000, Lee [26] proposed a binary encoding based on Genetic algorithm [9], [10] for knight's tour problem that was based on biological theories of evolution. In 2004, Hingston et al. [11] proposed an intelligent ant based algorithm which uses the fact that during ant's traversal of a graph, they deposit pheromones every time and doing so to enumerate admissible knight's tours in different sized chessboards. Parberry [16] also proposed a neural network based algorithm in which each admissible knight's move on the chessboard is represented by a neuron.

## V. Algorithms

Backtracking is a recursive algorithm approach. A backtracking algorithm proceeds to build a solution to any computational problem incrementally. Every time the algorithm needs to choose among different alternatives to the next phase of the solution, it recursively tries all possible options. The most simple and straightforward approach to this puzzle is backtracking algorithm, but naïve backtracking search is very slow, even very powerful computers need considerable time to produce output. With a clever heuristic, a backtrack algorithm may work faster. Warnsdorff, first turned this fact into a reality after devising a clever and simple heuristic. During construction of a knight's tour in a chessboard, at every stage, we are faced with some options and choose one option over the others. However, the option that we choose might lead to a dead end, then only we need to backtrack and choose some other alternate options.

Suppose, the knight is currently in square (a, b) and it chooses one of the eight possible moves to a square (if the square is not visited previously and the move is indeed a legal one). Then, we move the knight to that square and check recursively whether we can find a solution from that square. If the solution exists, then that square is marked as visited and again the knight chooses one of the eight possible moves and follows the same steps. If the next move is not possible, then the knight backtracks to the previous square (a', b') and tries out other possible alternatives. When all the squares are visited, we have found a sequence of knight moves which visits each square on the chessboard exactly once. Our algorithm computes one possible knight's tour for a chess board of a size 8×8.

```
findtour(a, b)
{ // a, b are integer and denotes chessboard's
// row, column  index
       if(!(a>-1 && a<8 && b>-1 && b<8 &&
arr[a][b]==0))
            return;
       point++;       // point variable is a
// global variable initialized to 0 and it
// maintains the indexing for two arrays x[]
// and y[]
       arr[a][b]=1;    // signify this square
// position is visited and hence flags it
       x[point]=a;
       y[point]=b;
       if(point==63)    //if the indexing of
// x[] and y[] array reaches the highest index
       printsol();
// Here are all possible eight moves that
// knight can take
       findtour (a-2,b+1);
       findtour (a-1,b+2);
       findtour (a+1,b+2);
       findtour (a+2,b+1);
       findtour (a+2,b-1);
       findtour (a+1,b-2);
       findtour (a-1,b-2);
       findtour (a-2,b-1);
// when the last call to findtour() does not
// satisfy, it comes back to the section below
       point--; // decrements point so that it
// may come back again to find a proper tour
       arr[a][b]=0;
}
```

This algorithm is invoked by initial call findtour(0,0). Execution time is also dependent on the choice of the starting square for the tour. Actually, our algorithm is backtracking paradigm, which is recursive in nature and is easier to implement. Here, variables 'a' and 'b' initially contain the starting square position of the knight. 'a' indicates row index and 'b' indicates column index. Next, within if condition, it checks whether the knight's position is within the bounds of the chessboard, one for rows other one for columns; also checks whether current square position has been visited before, 0 signifies square is unvisited 1 implies it's a visited square. 'arr[][]' is a 2D array of size 8×8. 'x[]' and 'y[]' are two 1D arrays, each of size 64. 'x[]' and 'y[]' actually stores the row and column indexes respectively for the 2D array to track each knight's move. printsol() function prints the entire knight's tour in 2D array one by one for each values stored in arrays 'x[]' and 'y[]'. Finally, 'arr[][]' is reassigned to 0 to make this square position unvisited so that it may come back again to find a proper tour. Our recursive backtracking algorithm is correct because it systematically explores all possible tours (solutions) and marks them.

## VI. Experimental evaluation

According to Knuth [13] - "One of the chief difficulties associated with the so-called backtracking technique for combinatorial problems has been our inability to predict the efficiency of a given algorithm, or to compare the efficiencies of different approaches, without actually writing and running the programs". The implementation poses challenging programming problems. Even though recursive backtracking algorithms generally take exponential time, its better if we evaluate our algorithm experimentally with practical data sets rather than trying to derive a worst case running time for it.
Here we have implemented our recursive backtracking search algorithm for the knight's tour puzzle.
Unlike Warnsdorff's heuristic which can only find closed knight's tours, our algorithm can find both open/closed tours. But for certain square positions, it takes time to progress for open or closed tours due to findtour() sequencing and starting knight's position.

Consider Fig. 2, here an open tour is found when the knight is starting from (0,0). When implemented our algorithm, the written program gives following output on 8×8 board.

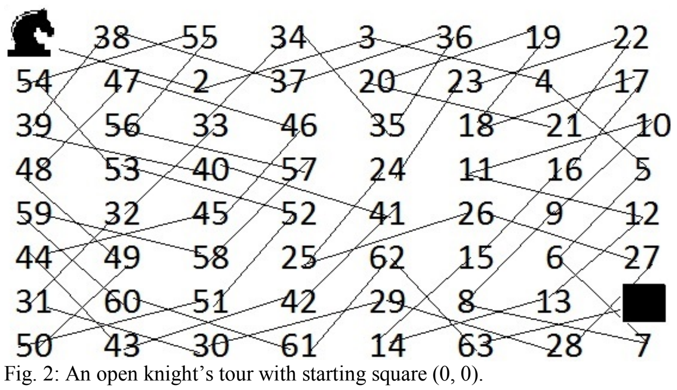

Fig. 2: An open knight's tour with starting square (0, 0).

As per our algorithm initially the knight is at square position (0,0), then it moves to square index (0+1,0+2) = (1,2). From (1,2) the knight moves to square index (1-1,2+2) = (0,4). From (0,4), the knight moves to square index (0+1,4+2) = (1,6). From (1,6), the knight moves to square index (1+2,6+1) = (3,7), and so on. The black square on square index (6, 7) ends the tour, which is clearly an open tour and is actually the 64th move of the knight.
Consider Fig. 3, here a closed tour is found when a knight is starting from index (5,6). Closed tour starts from index (5,6) and ends at index (5+2,6+1) = (7,7). Recall that a closed tour is that tour where starting and ending positions are one knight move away from each other. When implemented our algorithm, the written program gives following output on 8×8 board.

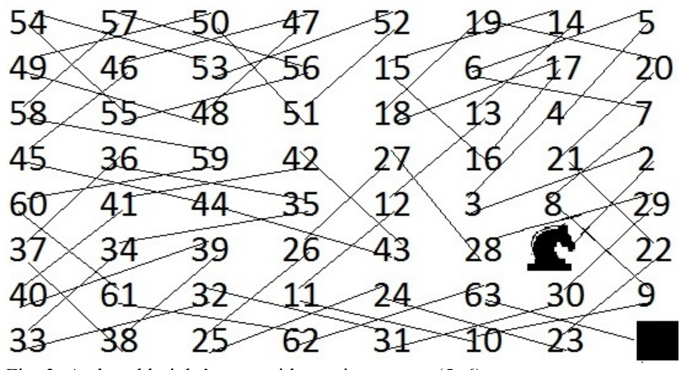

Fig. 3: A closed knight's tour with starting square (5, 6).

Our algorithm is made to find one complete closed/open tour. However if we wish to find all the closed tours then slight modification is needed in the printsol() function. Our algorithm gets an open tour starting from index (0,0) however it might also get a closed tour, if we do not stop our algorithm after printsol() gets executed once (we are stopping so that we can compare it with the other algorithms). But the time taken for its execution will be a lot to track.

Experiment shows that our proposed algorithm lies somewhere midway between Neural Networks and Ant Colony Enumeration on one side; and Warnsdorff's heuristic and Ira Pohl's extension of Warnsdorff's tie breaking on the other side. Hence our algorithm is faster than Ant Colony Enumeration and Neural Networks, but slower than Warnsdorff's heuristics. Our recursive backtracking algorithm is simple to understand and easy to implement. We did not implement Genetic algorithm as it was proved to be inexact because it is highly random in nature. Genetic algorithm [9] is slower than that of our algorithm. We also see that Neural Network's plot is outside the bounds of the graph which depicts that it is much slower and it took us 2.55 minutes to process which is beyond the range (0.5 sec) of the graph.

We abide by following naming convention for the rest of the paper.

| **Algorithm** | **Algorithm Name** |
|---|---|
| Algorithm 1 | Warnsdorff's Heuristic |
| Algorithm 2 | Ira Pohl's extension of Warnsdorff's tie breaking |
| Algorithm 3 | Our proposed Algorithm |
| Algorithm 4 | Ant Colony Enumeration (ACE) |
| Algorithm 5 | Neural Networks Algorithm |

For efficient comparison and experimental purposes, we kept all input sizes same i.e., 8×8 chessboard and we have synchronized all algorithms (Algorithm 1 to 5) starting from index (0, 0). Consider the following graph, here we plot algorithms 1,2,3,4,5 on the Y axis and Running Time (in seconds) on the X axis. There exists a curve which is actually an exponential line (dotted line). Our Algorithm 3 is situated below the exponential curve along with Algorithm 1 and Algorithm 2 which suggests that our proposed algorithm is not exactly as exponential as we think in terms of execution time. Whereas Algorithm 4 and Algorithm 5 appears above the exponential curve which proves them to be exponential or maybe worse.

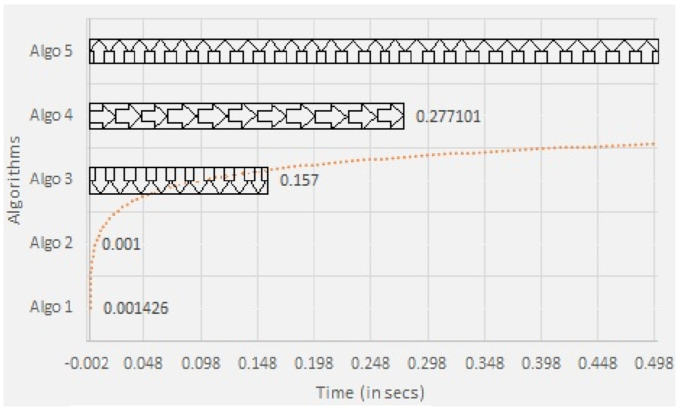

Fig. 4: Run Time Analysis on an 8×8 chessboard.

Also, our algorithm is more efficient than Algorithm 5 i.e., Neural Networks and Algorithm 4 i.e., Ant Colony Enumeration, which has been proved in figure 4. However, Algorithm 1 i.e., Warnsdorff's Heuristic and Algorithm 2 i.e., Ira Pohl's tie break application on Warnsdorff's is more efficient than Algorithm 3 i.e., our algorithm. But we can't entirely rely on Algorithm 1 or Algorithm 2 as those are heuristic i.e., guaranteed algorithmic solutions does not exist and can't be proved. Moreover, Algorithm 1 fails to give correct tours on chess board sizes above 49 × 49. Heuristics are "educated guesses" and will not give true results in some conditions. It is true that if we increase the chess board size, all algorithms (Algorithm 1 to 5) including our Algorithm 3 will take more time to process. Here, as the number of tours is huge, ACE algorithm is unable to list them all. Hence it can do no better than to run the algorithms for a certain number of attempted tours or for a certain amount of time, and examine the performance up to that point [11]. On 8 × 8 boards, Warnsdorff's rule was proved to be better than ACE algorithm. Whereas our algorithm is much faster than Ant Colony Enumeration, our Algorithm 3 takes less than exponential time as opposed to Algorithm 5 i.e., ACE algorithm which takes more than exponential time.

All the algorithms were implemented in C, and the run-time (execution time) was measured using the C library function, clock_t clock(void). To get the number of clock ticks in seconds, it was hence divided by the macro CLOCKS_PER_SEC. All the algorithms were fed with the same input size i.e., 8×8 chessboard with the knight starting from the first square position (0,0). The graph was plotted thereafter. The performance evaluation was simulated on a Intel(R) Core(TM) i5 2.30GHz CPU with 4.00 GB RAM, running Microsoft Windows 10 Home.

## VII. Conclusion

Example of the problems that can be attacked by backtracking algorithm does not mean that backtracking solution is the best approach for those problems. There may exist other algorithmic approach that gives optimal solution for those problems. Ant based algorithm and genetic algorithm are impractical as they execute for long time to produce solution of knight's tour. For standard 8×8 chessboard, our algorithm gives moderate time solution compared to other efficient solutions.

## VIII. Future work

For different sizes (irregular) of chessboards, rather than the regular 8 × 8, as well as rectangular boards, variations of the knight's tour problem. Starting from a given square, one can count the minimum number of moves required for the knight to finish its journey, i.e., to visit all other square only once. One may extend our algorithm for any size of n. One can plan to apply Best-First Search and A* search heuristics as the basis for faster solutions. People can also extend the same for higher dimensions.

## References

[1] I. Parberry, "An efficient algorithm for the Knight's tour problem", Discrete Applied Mathematics, vol. 73, 1997, pp. 251–260.
[2] I. Pohl, "A method for finding Hamiltonion paths and knight's tours", Communications of the ACM, vol. 10 (7), 1967, pp. 446–449.
[3] A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener, "Solution of the Knight's Hamiltonian Path Problem on Chessboards", Discrete Applied Mathematics, vol. 50 (2), 1994, pp. 125–134.
[4] O. Kyek, and I. Wegener, "Bounds on the Number of Knight's Tours", Discrete Applied Mathematics, vol. 74, 1997, pp. 171–181.
[5] H. C. Warnsdorff, "Des Rösselsprunges einfachste und allgemeinste Lösung. Schmalkalden", 1823.
[6] D. Squirrel, and P. Cull, "A Warnsdoff Rule Algorithm for Knight's Tours", 1996.
[7] M. Garey, and D. Johnson, *Computers and intractability: A guide to the Theory of NP-Completeness*, W. H. Freeman and company, 1979.
[8] S. Ganzfried, "A New Algorithm For Knight's Tours, Technical report", Oregon State University, 2004.
[9] J. Gharaibeh, Z. Qawagneh, and H. Zahawi, "Genetic Algorithms with Heuristic Knight's Tour Problem", GEM, 2007, pp. 177–181.
[10] V. S. Gordon, and T. J. Slocum, "The Knight's Tour – Evolutionary vs. Depth-First Search", in Proc. Congr. of Evolutionary Computation, 2004, pp. 1435–1440.
[11] P. Hingston, and G. Kendall, "Enumerating Knight's Tours using an Ant Colony Algorithm", Advances in Artificial Intelligence, The 18th Australian Joint Conf. on Artificial Intelligence, Cairns, Springer, 2004.
[12] H. J. R Murray, History of Chess, Clarendon Press, 1913.
[13] D. E. Knuth, "Estimating the efficiency of backtracking programs", Mathematics of computation, vol. 29 (129), 1975, pp. 121–136.
[14] D.E. Knuth, "Uncrossed knight's tours", Journal of Recreational Mathematics, vol 2, pp. 155–157, 1969.

[15] D. H. Lehmer, “Combinatorial problems with digital computers”, Proc. 4th Canadian Mathematics Congr., 1957, pp. 160–173.
[16] I. Parberry, Scalability of a Neural Network for the Knight’s Tour Problem, Tech. Report, Center for Research in Parallel and Distributed Computing, Dept. of Computer Science, University of North Texas, 1997.
[17] P. Cull, and J. De Curtins, “Knight’s tour revisited”, Fibonacci Quarterly, vol. 16, 1978, pp. 276–285.
[18] J.J. Watkins, Across the Board: The Mathematics of Chessboard Problems, Princeton University Press, 2004.
[19] H. R. Wiitala, The Knight’s Tour Problems on Boards with Holes, Tech. report, 1996.
[20] B. D. McKay, Knight’s Tours of an 8× 8 Chessboard, Tech. Report TR–CS–97–03 (Department of Computer Science, Australian National University), 1997.
[21] S. L. Marateck, How Good is the Warnsdorff’s Knight Heuristic?, New York University, 2008.
[22] M. Lobbing, and I. Wegener, “The Number of Knight’s Tours Equals 33,439,123,484,294 – Counting with Binary Decision Diagrams”, The Electronic Journal of Combinatorica, vol. 3(1), 1996.
[23] D. E. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms*, Addison-Wesley Professional, 2014.
[24] M. S. Petković, *Famous Puzzles of Great Mathematicians,* American, Mathematical Society, 2009.
[25] S. S. Lina, and C. L. Weib, “Optimal algorithms for constructing knight’s tours on arbitrary n×m chessboards”, Discrete Applied Mathematics, vol. 146, 2005, pp. 219 – 232.
[26] M. Lee, “Finding Solutions to the Knight’s Tour Problem Using Genetic Algorithms”, Genetic Algorithms and Genetic Programming at Stanford, Stanford Bookstore, 2000.
[27] L. Euler, “Solutio d’une Question Curieuse qui ne Peroit Soumise a Aucune Analyse”, Mem, Acad. Sci. Berlin, vol. 15, pp. 310–337, 1759.
[28] M. Gardner, “Problems that are built on the knight’s move in chess”, Scientific American, vol. 217 (4), 1967, pp. 128–132.
[29] L. Paris, “Heuristic strategies for the knight tour problem”, Proc. Int. Conf. on Artificial Intelligence, 2004, pp. 60–64.
[30] R. Cannon, and S. Dolan, “The Knight’s Tour”, The Mathematical Gazette, vol. 70(452), 1986, pp. 91–100.