Challenging the Record for the Maximum Solution to the Knight's Tour Problem

Masakatsu Okuno* Eiji Miyano**

(*Zoma Co., Ltd. **Kyushu Institute of Technology)

1 Introduction

Let $G = (V, E)$ be an undirected graph. A Hamilton cycle is a closed path that passes through each vertex in the vertex set V exactly once. The Hamilton cycle problem, which asks whether an input *graph G* has a Hamilton cycle, is one of the representative NP-complete problems [4]. However, because it is an important problem with many practical applications, many heuristic methods have been proposed. This paper deals with the Knight Tour Problem (KT problem) [1], **a** special case of the Hamilton cycle problem. The KT problem is a path search problem using chess, and it is a problem of finding a path for a knight that passes through all squares on an 8 × 8 chessboard. Furthermore, a more general problem of finding a knight path on a larger board has been considered [1].

The KT problem is an old puzzle, and with the advent of computers Before waiting for solutions, diverse solutions had been obtained manually. The primary focus was on pursuing beautiful solutions, and finding solutions for large boards manually was difficult. Subsequently, it has served as teaching material for backtracking algorithms based on computer depth-first search. However, the programs presented typically only manage to find solutions on 8 × 8 boards, with little evidence of attempts to tackle larger solutions. This research aims to develop methods for finding solutions on as large boards as possible, as quickly as possible. This paper presents two distinct heuristic methods concerning the order of visiting adjacent vertices at each vertex.

Consider the method. (1) The first is a method that makes the visit order of the eight vertices adjacent to each vertex identical.

(1) The first method involves using the same visitation order for all eight adjacent vertices.

(2) The second method uses a different visit order for each vertex. However, in either case, the visit order must be fixed before starting the path search. This paper considers the above two search orders and presents the results of their computer implementation. For the visit order in (1), knight's path can be found on boards up to 304 × 304 in size (specific paths will be shown at the presentation). For the visit order in (2), when the visit order for each individual vertex is chosen randomly, a knight's path can be found on boards up to 48 × 48 in size.

2 Proposed Algorithm

This paper considers an input unoriented *graph* G = (*V*, *E*), where *V* is a lattice board composed of an even × even number of vertices, and *E is* the set of edges connecting adjacent vertices that a knight can move to. We also consider an algorithm to find a path from a Start vertex to an End vertex in G, such that the path visits every vertex exactly once. Note that the algorithm proposed here works for any graph.

2.1 Basic Strategy of the Algorithm

The vertex degree (number of adjacent vertices) in the input graph is at most 8 before starting the algorithm. For path selection, (1) choose which adjacent vertex to move to, (2) move in depth-first order, and (3) backtrack if path selection fails. Furthermore, when moving from vertex *u* to vertex v, then to vertex *w*, all edges from vertex *v* to other vertices except edges uv and *vw* are removed from the graph. In other words, the fundamental principle of the path search algorithm is to move while reducing the number of candidate paths to the destination.

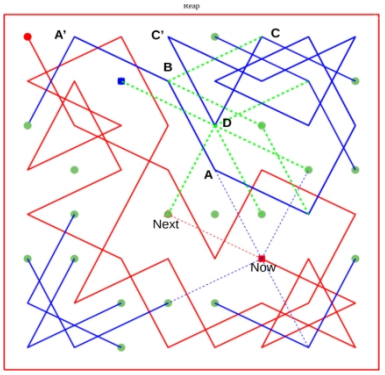

Figure 1: Example of algorithm operation on an 8 × 8 board

The proposed algorithm is based on backtracking with pruning, but further employs the following two main strategies. (**Adjacent Vertex Visit Order Strategy**) The order of visiting adjacent vertices at each vertex is determined at the start. We consider two methods: (1) using the same visit order for all vertices, and (2) using a different visit order for each vertex. (**Edge Deletion Strategy**) As described above, when visiting vertices in the order *u*, *v*, *w*, edges uv and *vw* connected to vertex *v* on the path are removed from the candidate set. Additionally, edges connected to vertices not on the path may sometimes be removed from the candidate set. Suppose there is a vertex *p* and two unvisited vertices *q* and *q′* connected to it, with *q* and q' having degree 2. If we move from *p* to a vertex other than *q* and q', the edges that could become path candidates for *q* and q' each become one, making it impossible to form a final Knight's Path. Therefore, even if the degree of *p* is 3 or higher, at this point, we can remove edges other than *pq* and pq'.

Here, we illustrate an example of the algorithm's operation (see Figure 1). The algorithm is fundamentally based on a depth-first backtracking method, discovering Dead Ends – path terminations— and performing pruning. In Figure 1, Start is represented by the red circle in the upper left corner, and End by the blue circle. The search determines whether to move from the current visited vertex (Now, red square) to the next candidate vertex (Next, red triangle) by checking the validity of the move. It then decides whether to actually move or to skip the move and examine the next candidate vertex. In Figure 1, attempting to move from Now to Next causes the edges connected to Now's adjacent vertices to be deleted. The blue dashed lines from Now represent these edge deletions. As a result, the degree of vertex A, which was 3, becomes 2. Since both vertices A and A' have a degree of 2, the intermediate vertex B can also be reduced to degree 2 using the aforementioned edge deletion strategy. That is, the remaining edge can be deleted. The deleted edge is shown as a green dashed line. Furthermore, the degree of C decreases from 3 to 2, and the intermediate vertex D between C and C' can also be reduced to degree 2 using the edge deletion strategy. This edge deletion is
A chain reaction occurs, deleting the seven green dashed edges. This increases the number of simple paths represented by blue lines without branches.

Since the order of visiting adjacent vertices on a path is unique, search efficiency increases exponentially. Edge deletion may sometimes create a Dead End. A Dead End is a state where no path can reach the end, regardless of the route chosen. The proposed algorithm checks three types of Dead Ends and performs backtracking in depth-first search. In Figure 2, the large blue circles indicate Dead

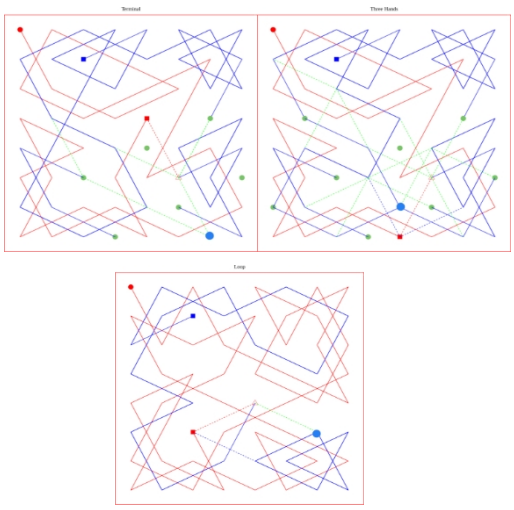

Figure 2: (Top left) Terminal, (Top right) Tree, (Bottom) Loop

End. (Terminal) Deleting an edge creates a vertex with degree 1, causing path search failure. (Three Hands) If three or more simple paths originate from a vertex, at least one simple path cannot be part of a Knight's Path. (Loop) If an isolated loop forms, the search can be terminated and backtracked.

Computer implementation experiments confirm that for boards around 10 × 10, dead ends do not occur and backtracking is rarely necessary. For 14 × 14 boards, backtracking occurs relatively early, enabling rapid discovery of knight paths.

2.2 Connected Component Strategy and Eulerian Graph Strategy

For any vertex subset $S \subseteq V$ of the graph $G = (V, E)$, Let $k(G - S)$ denote the number of connected components obtained by removing S from G. It is known [2] that a sufficient condition for G to have a Hamilton cycle is that $|S| \geq k\ (G - S)$. This represents the condition for moving between each connected component, and the proposed algorithm uses a pruning strategy motivated by this relationship. In addition, focusing on the entry and exit of connected components, we utilize a pruning strategy based on Euler's theorem [3], which is a condition for so-called one-stroke drawing. In this paper, we define **an island** as an induced subgraph in which all vertices have a degree of 3 or more. Furthermore, **a simple path** connecting two vertices in different islands is called a bridge. In Figure 3, the red line indicates a traversed path, while the green, blue, orange, and purple dots represent four distinct islands. The blue line represents a simple path. An island is a connected component and must satisfy the following conditions (I) to (III) for Next, End, the island itself, and the bridge.

(I) For all islands, the number of bridges must be even. In this case, Next and End must be on the same island.

(II) Suppose an island has an odd number of bridges. In this case, there are exactly two islands with an odd number of bridges. One island must contain Next and the other must contain End.

(III) There are no islands without bridges.

(I) This indicates that if you enter an island as Next, you cannot reach End unless the number of bridges is even. Furthermore, Next is the current Start. Each island must be exited after entering, and only the island containing the final End can be entered to terminate the Knight's path. (III) indicates that the island cannot be entered or exited, thus having no path.

The computational complexity of the algorithm increases, but by employing the above strategy at each stage and performing early pruning during back

Although the computational complexity of the algorithm increases, employing the above strategy at each stage enables early pruning during backtracking, leading to faster algorithm execution and pathfinding on large-scale boards.

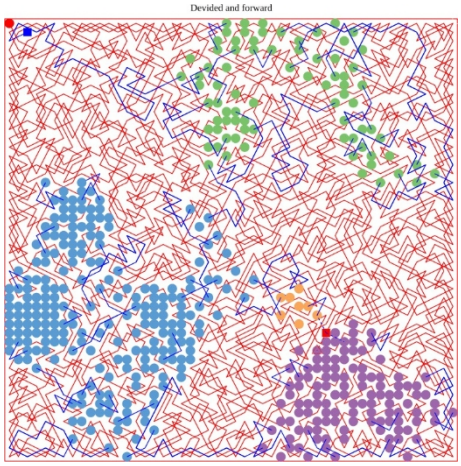

Figure 3: Islands and Bridges on a 48 × 48 Board

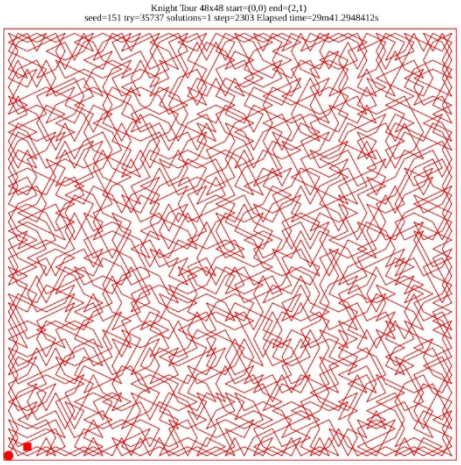

Figure 4: Knight's Path on a 48 × 48 Board

3 Acceleration Techniques and Implementation Results

In the implementation on the computer, we achieve acceleration through two parallelization and threading approaches. (1) The first involves performing sequential computation for moving from the start to several adjacent vertices, then deleting edges from the input graph based on the path selection at that point. Once a certain amount of path selection is complete, we switch to parallel path search using threading. Since sequential and parallel computation involve trade-offs, we attempt to find an efficient switching point. (2) The second approach first prepares all possible visit orders for adjacent vertices based on random numbers. We generate P such random-based orders. Next, we prepare a thread for each of the P orders and perform fully asynchronous parallel exploration. Solutions for 48 × 48 and 304 × 304 were obtained using these speed-up techniques (see Figure 4).

Acknowledgments. This research was supported by Grant-in-Aid for Scientific Research JP24K02902.

References

[1] Knight's tour notes (compiled by George Jelliss). https://www.mayhematics.com/t/t.htm

[2] V. Chvatal. Tough graphs and Hamiltonian circuits. Discrete Math., vol.5, pp.215 – 228 (1973)

[3] L. Euler. Solution of a problem pertaining to geometry situs. Comment. Acad. Sci. I. Petropolitanae, 8, 128–140 (1736)

[4] M.R. Garey and D.S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman (1979)