2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics
Suntec Convention and Exhibition Center
Singapore, July 14-17, 2009

Transportation Network Navigation with Turn Penalties

Gene Eu Jan
Member, *IEEE*
Institute of Electrical Engineering
National Taipei University
Sun Shia, Taipei, Taiwan
gejan@mail.ntpu.edu.tw

Ming Che Lee
Department of Computer and Communication Engineering,
Ming Chuan University
Taoyuan, Taiwan
leemc@mcu.edu.tw

S. G. Hsieh
Department of Electrical Engineering
National Taiwan Ocean University
Keelung, Taiwan
sghsieh@mail.ntou.edu.tw

Yung-Yuan Chen
Department of Computer Science
Chung Hwa University,
Hsin-Chu, Taiwan
chenyy@chu.edu.tw

Abstract— **The path length and number of turns are the major factors in path planning of transportation and navigation systems. Shortest path planning has been widely studied in the literatures. Most researches only take the issue of shortest distance into account, and the impact of turns are rarely mentioned, that is, the shortest path may not be the fastest. Considering both two factors in a path-searching algorithm is NP-complete. This paper proposes two algorithms: the Least-Turn Path Algorithm and the Minimum-Cost Path Algorithm to balance both the path length and turns. The proposed algorithms adapt Lee's rectilinear routing algorithm to find a least-turn path with turn penalty on a mesh-connected network. In addition, Kirby's concept and a modified Dijkstra's algorithm are also introduced for the proposed minimum-cost path algorithm, which considers the turn penalty and the length factor on a transportation network. The time complexities for both algorithms are *O*(*N*), where *N* is the number of nodes on a mesh-connected network or the intersections on a transportation network.**

Key Words: Shortest path, Dijkstra's algorithm, Lee's algorithm, NP-Complete, transportation networks, turn penalties.

I. Introduction

PLANNING a connected path from a source node to a destination node on a mesh-connected network under certain criteria is an important research subject, and has been widely applied to Printing Circuit Board (PCB) wiring, Integrated Circuit (IC) layout and the route-searching for transportation vehicles and robots in the Geographical Information System (GIS). Researches for transportation vehicles assume that the vehicles are under a constant speed, and most of them try to find a shortest path with a least amount of time. However, due to inertia and safety, the vehicles need to slow down and brake hard before the corner and this will increase the total time. Thus, the number of turns is also an important factor and cannot be overlooked in the path planning.

The background of this article starts from Lee's research on finding a shortest path in a raster space, and the path connection algorithm for searching the shortest path is still being widely studied [1].

The concept of turn penalty in path planning was first introduced by Caldwell [2], which established a pseudo-network to show the relationship between the nodes on a transportation network. Martin [3] employed other method to detect the occurrence of a turn via the variants of a constant. Kirby [4] dealt with the "The Mixed Rural Postman Problem with Turn Penalties (MRPPTP)" and a polynomial transformation was proposed to translate the problem into the "Asymmetric Traveling Salesman Problem (ATSP)". Frank [5] proved that the time complexity of these algorithms presented by Caldwell, Kirby and Potts were too large. Very recently, many turn cost related researches were proposed [6][7].

In this article, two path planning algorithms are proposed. The first one is try to find a path with minimum turns from the source node to the destination node on a mesh-connected network. The idea of this algorithm is based on Lee's rectilinear routing concept. The second algorithm applies Kirby's concept and a modified Dijkstra's algorithm. We employ the accumulative concept to define the cost of a path, reestablish a two-dimensional network, and construct the minimum-cost path under turn penalty and length factor. This algorithm can be used for the mesh-connected networks and transportation networks. The time complexity for both algorithms is *O*(*N*) where *N* is the number of nodes.

The contents of this paper are as follows: Section II introduces the space structure and Lee's algorithm. The least- turn path algorithm and the minimum-cost path algorithm are presented in Section III and Section IV, respectively. Section V concludes the paper.

II. Space Structure and Lee's Structure

A. Raster Graph

A raster is a graph that can be represented by rectangular, triangular, and hexangular cells. Our approach adapts the first model whereas the rectangular cells can be further divided into smaller cells recursively.

The cell map indicates which cells constitute obstacles at the very least. As the algorithm unfolds, the cell map can also be used to indicate the intermediate status of the cells. Lee's algorithm considers four directions of each central cell,

that is, the above, below, to the left, and to the right of a cell, and all of the cells are 1 unit distance from the center cell. Each cell is treated as a basic unit of the iteration in the algorithm. The source cell is assigned to loop 0 (zero) and its neighbor cells are assigned to loop 1 (one). Continuously, the set of cells in (i+1)-th loop can be recursively defined as the set of neighbor cells of the ith loop cells besides the cells belong to the (i-1)-th loop.

In this paper, the raster graph is used as the space structure. To simplify the description, each cell is replaced by a node and two adjacent nodes are connected by a link.

B. Lee's Algorithm

Lee's shortest path connection algorithm proposed in 1961 is the most commonly used algorithm for finding a path between two vertices on a planar rectangular grid. The key to the popularity of this algorithm is its simplicity and its guarantee of finding an optimal solution if it exists. The time complexity is $O(N)$, where N is the number of vertices on the grid. Lee's algorithm can solve the following problems effectively.

(1) Finding a path with shortest distance.
(2) Finding a path that bypasses all obstacles.

Lee's algorithm works in C-space which is usually represented by rectangular cells in 2-D planes. Although the proposed algorithm is not restricted to the rectangular configuration, cells such as triangular or hexangular cells are not easy to display space structure in a computerized graph. Rectangular cell maps are thus traditionally used for C-space.

This algorithm first generates a "search wave" that starts from the source cell (designated as loop 0, stored in L list), propagates to its neighbor cells (designated as loop 1), and proceeds continuously until the destination cell is reached or the wave is unable to propagate further. The adjacent cells of the search wave are called frontier cells (stored in L1 list). A cell cannot be added to the set of frontier cells unless it becomes the neighbor of a frontier cell. In addition, a cell cannot be removed from the set of frontier cells until all its neighbors that can be and have become frontier cells. Lee's algorithm has a distinct property: a cell is reached while it becomes a frontier cell.

In short, this algorithm can be divided into three phases:

(a) Wave propagation phase (Exploration phase)
Starting from the source node, wavefronts are propagated to neighbor nodes until the destination node is reached.

(b) Backtracking phase
A path is backtracked from the destination node to the source node.

(c) Reversing phase
The shortest path is obtained by reversing all the edges of the path.

III. The Least-Turn Algorithm

This section illustrates the least- turn algorithm which generates the path with minimum turns on a mesh-connected network. The proposed algorithm adjusts some logical factors from Lee's algorithm in the wave propagation phase. The purpose is to decide whether to turn or not.

TABLE I
NOTATION USED IN THE LEAST-TURN ALGORITHM

Notation	Meaning
(i,j)	The index of a cell in the cell map.
$C_{i,j}$	Cell with index (i,j).
$Turn_{i,j}$	Records the number of turns from the source node to $C_{i,j}$, the initial value is ∞.
Wave	Linked list that stores the indices of the cells to be processed.

A. Data Structure

The size of the mesh-connected network in the proposed model is assumed to $m \times n$ with total number of nodes N and the length between any two neighbors is one unit. As shown in Table I, a node with indices (i, j) in the mesh-connected network is denoted by $C_{i,j.}$ and its neighbors at North, East, South, and West side is denoted by $C_{i,j+1}$, $C_{i+1,j}$, $C_{i,j-1}$, and $C_{i-1,j}$, respectively.

The algorithm first evaluates the number of turns of $C_{i,j}$'s four neighbors and thus $C_{i,j.}$ is designated as the predecessor of these nodes. There may be more than one predecessor for each node since there may be different paths that come from different directions.

In the backtracking phase, the directions of $C_{i,j.}$'s predecessors are recorded in four specific designed Boolean variables: *North*, *East*, *South*, and *West*, and the initial values are all set to false. Furthermore, two linked lists, *wave* and *new_ wave*, are used to store temporary indices of the cells to be processed.

B. The Least-Turn Path Algorithm

There are three major steps in the proposed algorithm. Step 1: The initial value of $Turn_{i,j}$ of the source node S is set to 0 (zero), which represents that there's no need to make a turn to reach this node. Step 2: Computing the number of turns $Turn_{i,j}$ for each node by invoking the wave-propagating procedure. Step 3: Backtracking the path with least number of turns.

In step 1, the source node is added to wave loop 0 (zero) and stored in the linked list, *wave*. In step 2, each node's neighbors (called p_i) are inspected. If there is no need to make a turn to reach p_i, p_i is moved into wave loop 0, otherwise p_i is moved into wave loop 1, stored in *new_ wave* list and records the direction of its predecessor. Continuously, the wave propagates through the nodes until it cannot propagate anymore. That is, all the nodes needed to make a turn to reach the nodes in the ith loop are inserted to loop (i+1)-th. Those nodes that can be reached without turning remain in the ith loop. Once all the (i+1)-th loop nodes reached by the ith loop nodes are added to the *new_*

wave list, these *i*th loop nodes in the *wave* list are deleted. In step 3, according to the direction of the predecessor recorded in step 2, we can trace back step by step to obtain the least-turn path.

For the convenience of understanding the concept of this algorithm and the following presented algorithm, two functions, *Turns_Counting* and *Update* are introduced in addition. The function *Turns_Counting* is to determine whether a node $C_{i,j}$ needs to make a turn to reach its neighbors and then records the number of turns.

In summary, Step 1 sets the source node as loop 0 of the search wave. Step 2 calls the function *Turns_Counting* to calculate the number of turns to reach its neighbor nodes one by one. If the number of turns is smaller, function *Update* is called to update *Turns_Counting* to the smaller number. If the number of turns is equal to the previously stored number in the neighbor node, it will not be updated and the related path information is recorded. Continuously, the algorithm expands the search wave until it cannot expand any further. Step 3 traces back to obtain the least-turn path based on the path information recorded in Step 2. If there are multiple choices, the one with no change in original direction will be chosen.

Figure 1 shows an example of the least-turn path algorithm on the mesh-connected network.

IV. Correctness and Time Complexity of The Least-Turn Path Algorithm

A. Correctness

There are three steps in the least-turn path algorithm.

Step 1: Initialization, setting the initial value of number of turns $Turn_{i,j}$ of source node to 0 (zero). This means that we don't need to make a turn to reach this node.

TABLE II

The Function Of Turns_counting

Function 1: Turns_Counting ($C_{i,j}$, neighbor)	
Step 1.	Record the number of turns needed from $C_{i,j}$ to the neighbor

TABLE III

The Function Of Update

Function 2: Update ($C_{i,j}$, neighbor)	
Step 1:	Based on the result obtained by *Turns_Counting*, update the value of turn numbers of this neighbor node.
Step 2:	Delete the information related to the neighbor's predecessor and assign $C_{i,j}$ to the new predecessor.
Step 3:	If the number of turns does not increase, add this neighbor node to the *wave* list.
Step 4:	If the number of turns increases, add this neighbor node to the *new_ wave* list.

Step 2; Compute the number of turns $Turn_{i,j}$ for each node by using wave propagating procedure. The following proves the correctness by the apagoge method.

When we remove node $C_{i,j}$ from the *k*th linked list, $C_{i,j}$ has already stored the least number of turns $Turn_{i,j}$ from the source node. For any node $C_{i,j}$, if $Turn_{i,j} = k$, it means that this node is recorded in the *k*th linked list; and $0 \le k \le Turn_{max}$, where $Turn_{max}$ is the maximum value of least number of turns from source node to any node. Assuming

Algorithm 1. The Least-Turn Path Algorithm	
Step 1:	Initialization Set the initial value of $Turn_{i,j}$ of source node to 0 (zero) and store its indices in the *new_ wave* list.
Step 2:	Wave Propagation
Step 2.1:	*wave* ← *new_wave* *new_wave* ← ψ
Step 2.2	Remove the node $C_{i,j}$ from *wave*.
Step 2.3	Call Turns_Counting ($C_{i,j}$, neighbor) to process $C_{i,j}$'s neighbors one by one excepting the predecessor of $C_{i,j}$.
Step 2.4	If Turns_Counting ($C_{i,j}$, neighbor) obtains a smaller number of turns, call Update ($C_{i,j}$, neighbor).
Step 2.5	If Turns_Counting ($C_{i,j}$, neighbor) equals to the turns and $C_{i,j}$ is also a predecessor of this neighbor node, this information is recorded in the Flag of this neighbor node.
Step 2.6	If the *wave* list is not empty, return to Step 2.2.
Step 2.7	If the *new_wave* list is not empty, return to Step 2.1.
Step 3	Back Tracking Starting from the destination node. This step traces back to the next qualified node based on the previously recorded predecessor until the source node is reached. If there are several predecessors exist, choose the one in the same direction of the path.

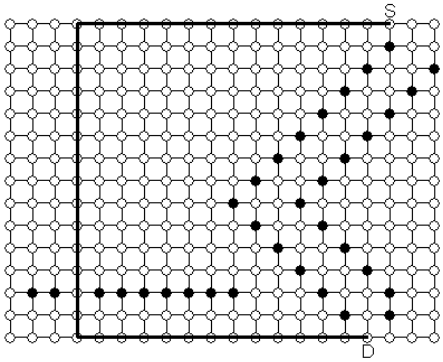

Fig. 1. The path obtained by the least-turn path algorithm.

that there exists a smaller least number of turns $Turn_{i,j}$' where $Turn_{i,j}$'< $Turn_{i,j}$ and $Turn_{i,j}$' was calculated from another cell $C_{i',j'}$. We then have to prove that $Turn_{i,j}$' is equal to $Turn_{i,j}$ or $Turn_{i,j}$' does not exist.

Suppose that node $C_{i',j'}$ is recorded in the mth linked list where $0 \le m \le k$-1. According to Steps 2.3 and 2.4, the number of turns $Turn_{i,j}$ should be replaced by $Turn_{i,j}$' already and thus the value of $Turn_{i,j}$' is that of $Turn_{i,j}$.

In addition, suppose that node $C_{i',j'}$ is recorded in the current kth linked list or in the nth linked list where $k \le n \le Turn_{max}$ and $Turn_{i,j} = k \le Turn_{i',j'}$. Since $Turn_{i,j}' = Turn_{i',j'} + trun_{i,j}^{i',j'}$ where $trun_{i,j}^{i',j'}$ is the number of turns needed for node $C_{i',j'}$ to enter node $C_{i,j}$. According to the definition, $C_{i',j'}$ and $C_{i,j}$ are adjacent nodes. Moreover, since the value of $trun_{i,j}^{i',j'}$ is either 0 or 1, we can obtain that $Turn_{i,j} \le Turn_{i,j}'$. This contradicts the original assumption. Thus, $Turn_{i,j}$' does not exist.

Once step 2 is executed, we can assure that the number of turns recorded in each node is minimum from the source node to it. In step 3, back tracking is used to find a way from the destination node to the source node step by step. That is, this step continues finding a suitable node which is recorded as the predecessor in step 1 until the source node is reached. If there is more than one predecessor, the one that does not change the original direction of the existing path is chosen and thus results in avoiding extra turns to get truly the least-turn path.

B. Time Complexity of the Least-Turn Path Algorithm

The initialization step costs constant running time. In step 2, since there are at most N nodes to be removed from the linked list for computation with their neighbors and they will not be put back for iterations after processing, thus the time complexity is $O(N)$. Step 3 needs N times computation in the worst case to trace back to get the least-turn path. Hence, the time complexity for this step is also $O(N)$. Overall, the time complexity for the least-turn path algorithm is $O(N)$, where N is the total number of nodes in the network.

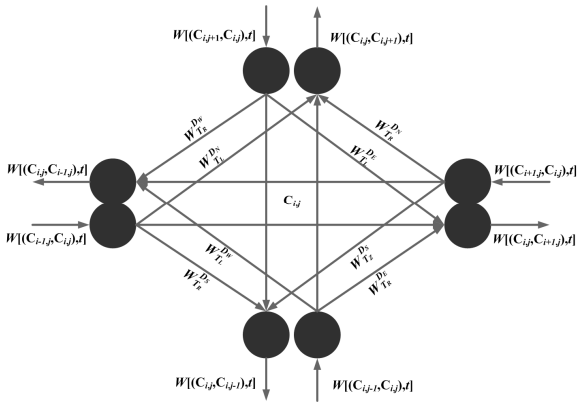

Fig. 2. Node structure of the minimum-cost path algorithm.

V. Minimal-Cost Path Algorithm

This section presents a minimal-cost path algorithm that balancing the path length and the turn factor.

Considering both two factors in a path-searching algorithm is NP- complete [8]. Our method adapts Kirby's concept [4] and modifies the data structure such that each node is transformed into a pseudo-network consisting of several pseudo nodes and links. The pseudo-network can express turning and go- straight status [4] and the Dijkra's algorithm [9] is applied to find a path with minimum cost.

A. Data Structure

This section introduces Kirby's concept. Caldwell first introduced the least-turn path problem [2] and many researchers studied this problem subsequently [3,4,5]. As shown in Fig. 2, each node in Kirby's model is replaced by 8 pseudo nodes in a pseudo-network and these nodes are connected by links. These links represent the status of movements, which including length and direction of the movement. The weighting factor of cost for each movement is expressed by $W_{T_h}^{D_m}(C_{i,j},t)$ which can be defined as a constant value or function of time. In the factor, D_m is the incoming direction and can be expressed by $D_m = \{(D_0, D_1, D_2, D_3) \mid (D_N, D_E, D_S, D_W)\}$, which represents the directions toward North, East, South, and West respectively.

The T_h represents the type of movement and $T_h = \{(T_0, T_1, T_2) \mid (T_L, T_R, T_S)\}$, which represents Left turn, Right turn and Straight-going, respectively. To simplify this problem, we use a constant to define $W_{T_h}^{D_m}(C_{i,j},t)$ in this article.

Kirby also proposed the concept of node sharing in which the pseudo nodes shares with neighbor nodes. When the pseudo nodes of a node as shown in Fig. 3 share pseudo nodes with its neighbors, the cost of movement from node to node is to be accumulated to the three links of the corresponding pseudo node. Differentiating the previously defined weighting factor $W_{T_h}^{D_m}(C_{i,j},t)$, we can define the new weighting factor as $SW_{T_h}^{D_m}$ as

$$SW_{T_h}^{D_m} = \Sigma W_{T_h}^{D_m}(C_{i,j},t) + W[(C_{i,j},C_{i',j'}),t] \quad (1)$$

where $i' = i + (+1)^E + (-1)^W$, $j' = j + (+1)^N + (-1)^S$, and E, W, N, S represents the Eastern, Western, Northern and Southern neighbors of $C_{i,j}$, respectively. $W[(C_{i,j},C_{i',j'}),t]$ is the cost needed for movement from node $C_{i,j}$ to node $C_{i',j'}$ and is defined as a constant value or function of time. For simplicity, we define the $W[(C_{i,j},C_{i',j'}),t]$ as a constant in this article.

The cost of the path is the summation of the cost on the link where the path goes through. Suppose that the path goes through the nodes $C_{i(0),j(0)\ i(1),j(1)\ i(2),j(2)\ldots.\ i(d),j(d)}$ from source node to any node $C_{i,j}$ sequentially, where $i(0), j(0)\ i(1), j(1)\ i(2), j(2)\ldots.\ i(d), j(d)$ are indices of the nodes the path goes through. Let $Cost_s^d$ be the cost of the path, thus

$$Cost_s^d = \sum_{l=0}^{d} SW_{T_h}^{D_m}(C_{i(l),j(l)},t) \quad (2)$$

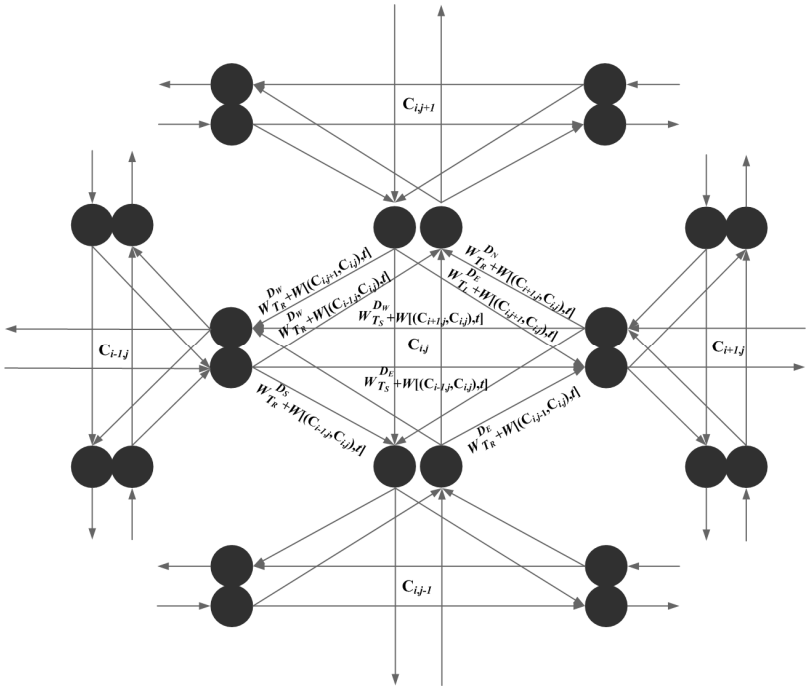

Fig. 3. Node structure after sharing pseudo nodes.'

Kirby proposed such a complete pseudo-network to simulate the movement behavior among nodes and constructed a tree-type searching algorithm for the least-turn path algorithm. However, the time complexity is too large.

A. *Dijkstra's Shortest Path Algorithm*

Dijkstra proposed a shortest path algorithm for finding a minimum-cost path in a weighted directed graph in 1959. The proposed algorithm can be applied to any weighted directed graph with non-negative link weight. The main concept is to maintain a set S, which includes all the nodes that have been found with a shortest path from the source node s to the destination node d. All other nodes that does not belong to the set S are included in a set named Q, that is, $Q = \{u \mid u \in V–S\}$. It then searches for the minimum-cost node in set Q and inserts the node to the set S. This procedure executes continuously until the destination is reached.

Let $G = (V, E)$ be a directed graph where V and E are the numbers of nodes and links in G respectively, and assume that the weight or the cost of each link $(u,v) \in E$, $w(u,v) \geq 0$, the Dijkstra's algorithm can be described as follows:

Dijkstra applies a linear sorting to find the node with smallest value in set Q which to be added to set S, therefore the time complexity is $O(V^2+E)$, where V is the number of nodes and E is the number of links. Many related researches have been published afterwards and most of them concentrate on improving the sorting strategy with smallest running time. The method proposed by Ahuja [10] is the one with shortest running time.

Ahuja applies both the Radix heap and the Fibonacci heap and the time complexity is $O(E + V\sqrt{\log W_{max}})$, where W_{max} is the maximum cost or weight factor of the links.

Algorithm：Dijkstra(G,w,s)
Initialize-Single-Source(G,s) $S \leftarrow \psi$ $Q \leftarrow V[G]$ While $Q \neq \psi$ do u ← Extract-Min(Q) $S \leftarrow S \cup \{u\}$ for each node $v \in Adj[u]$ do Relax(u,v,w)

B. *The Minimum-Cost Path Algorithm*

This section introduces the minimal-cost path algorithm. There are four steps in the proposed algorithm. The first one is the initialization step. We adapt Kirby's concept to rebuild the network and store the pseudo nodes of the source node into set S. Step 2 invokes the Dijkstra's algorithm and expands the set S continuously until the minimum-cost path from the source node to any node has been found for all nodes. Step 3 selects the pseudo node of the destination node with minimum cost $SW_{T_h}^{D_m}$ and the path it represented is the minimum-cost path from the source node to the destination node. Step 4 presents the back-tracking procedure. This step starts from the destination node and ends at the source node to find the minimum-cost path via the information of the predecessor recorded in Steps 2 and 3.

This minimum-cost path algorithm is summarized as follows:

Algorithm 2: The Minimum-Cost Path Planning Algorithm	
Step 1:	Initialization
Step 1.1:	Adopt Kirby's concept to rebuild the network structure.
Step 1.2:	Store the pseudo nodes of the source node to set *S*.
Step 2	Evaluate the cost for each pseudo node.
Step 2.1	Execute the modified Dijkstra's algorithm continuously until all the pseudo nodes on the mesh-connected network have been added to the set *S*.
Step 3	Find the minimum-cost pseudo node of the destination node.
Step 3.1	Select the pseudo node with minimum cost from the pseudo nodes of the destination node.
Step 4	Back Tracking. Start from the selected pseudo node within the destination node by using the data recorded in Steps 2 and 3 and then backtrack to the source node. After that, the path with minimum cost is found.

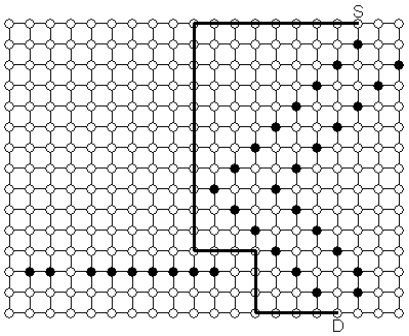

Fig. 4. Experimental result of the minimum-cost path algorithm.

Fig. 4 illustrates an experimental result of the proposed minimal-cost path algorithm. The cost of the left turn and right turn is the same and is assumed as 6 times of the straightforward direction. The result shows that the total path length is 29 units with 4 turns.

VI. Correctness and the Time Complexity of the Minimum-Cost Path Algorithm

A. Correctness of the Minimum-Cost Path Planning Algorithm

This algorithm adapts Kirby's concept by using a pseudo network structure to replace all the nodes on a mesh-connected network and translates the network into a direct-connected graph. After that, the modified Dijkstra's algorithm is used to find the minimum-cost path. The correctness of the proposed algorithm is therefore inherited from Dijkstra's algorithm.

B. The Time Complexity of Minimum-Cost Path Algorithm

Assuming that the mesh-connected network is sized $m \times n$ with N of nodes in our algorithm. Each node is composed by a pseudo-network of 4 pseudo nodes and 12 links. The initialization step costs a constant running time. Step 2 modifies Dijkstra's algorithm to find the shortest path and the time complexity is known as $O(E + V\sqrt{logW_{max}})$.

There are total $12N$ links and $4N$ pseudo nodes in the network and W_{max} is a constant, the time complexity of this step is $O(N)$. In Step 3 for the destination node, the algorithm selects the minimum-cost path from its 4 pseudo nodes and the time is also a constant. The backtracking step needs no more than $2N$ computations to obtain the path in the worse case. Therefore, the time complexity of the minimum-cost path planning algorithm is $O(N)$.

VII. Discussion and Conclusions

This paper proposed two path planning algorithms for transportation networks - the least-turn path algorithm and the minimum-cost path algorithm, which adopt Lee's rectilinear routing algorithm and Kirby's concept. Shortest path planning has been widely studied in the literatures, and is applied to many applications, such as the satellite navigation. The proposed minimum-cost path algorithm evaluated the cost of the path and defined the parameters for different requirements to find the shortest path, the least-turn path, the shortest path with least turns, the least-turn path with shortest length, and the optimum path with compromised length and number of turns. The proposed algorithms are efficient and only cost $O(N)$ time complexity.

References

[1] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Trans. Electron. Computer*, vol. EC-10, pp. 346- 365, Sept. 1961.

[2] T. Caldwell, "On finding minimum routes in a network with turn penalties," *Comm. of ACM*, 4, pp. 107-108, 1961.

[3] B. V. Martin, F. W. Memmott, and A. J. Bone, "Principles and techniques of predicting future demand for urban area transportation," *M.I.T. Rep*. No.3, 1961.

[4] R. F. Kirby and R. B. Potts, "The Minimum Route Problem for Networks with Turn Penalties and Prohibitions," *Transportation Research*, vol.3, pp.397-408, 1969.

[5] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. Computers*, vol. 9, pp. 907-914, Sept. 1974.

[6] E. M. Arkin , M. A. Bender, E. D. Demaine, S. P. Fekete,J. S. B. Mitchell, S. Sethia, "Optimal covering tours with turn costs", in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms,* pp. 138-147, Washington, D.C, 2001.

[7] G. L. Clossey, and P. Soriano, "Solving arc routing problems with turn penalties." *Technical Report G-2000-05*, Le Groupe d'etudes et de recherche en analyse des decisions, Montreal, Canada, 2000.

[8] M. R. Garey and D. S. Johnson, Computer and Intractability: A Guide to the Theory of NP-Completeness, *W. H. Freeman*, 1979.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 508-517, 1993.

[10] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of ACM,* vol. 37, pp. 213-223, 1990.