Paper

Non-crossing Rectilinear Shortest Minimum Bend Paths in the Presence of Rectilinear Obstacles

Shylashree Nagaraja

Department of Electronics & Communication Engineering, Rastreeya Vidyalaya College of Engineering, Bengaluru, Karnataka, India

https://doi.org/10.26636/jtit.2018.120417

Abstract—The paper presents a new algorithm to determine the shortest, non-crossing, rectilinear paths in a two-dimensional grid graph. The shortest paths are determined in a manner ensuring that they do not cross each other and bypass any obstacles present. Such shortest paths are applied in robotic chip design, suburban railway track layouts, routing traffic in wireless sensor networks, printed circuit board design routing, etc. When more than one equal length non-crossing path is present between the source and the destination, the proposed algorithm selects the path which has the least number of corners (bends) along the path. This feature makes the path more suitable for moving objects, such as unmanned vehicles. In the author's scheme presented herein, the grid points are the vertices of the graph and the lines joining the grid points are the edges of the graph. The obstacles are represented by their boundary grid points. Once the graph is ready, an adjacency matrix is generated and the Floyd-Warshall all-pairs shortest path algorithm is used iteratively to identify the non-crossing shortest paths. To get the minimum number of bends in a path, we make a modification to the Floyd-Warshall algorithm, which is constitutes the main contribution of the author presented herein.

Keywords—Floyd-Warshall algorithm, rectilinear non-crossing shortest paths, rectilinear obstacles.

1. Introduction

The problem of non-crossing shortest paths is well researched and several algorithms are described in literature, i.e. [1]–[3]. In general, the shortest paths reduce the time and cost of communication. Non-crossing paths are essential in VLSI chip track layouts, printed circuit board routing, robotic motion control [4], [5], etc.

Obstacles are natural and common in graphs representing geometrical/geographical scenarios In a printed circuit board layout, the components act as obstacles for the routing paths. Similarly, in transportation layouts, obstacles and prohibited zones are very common. In such a situation, we have to find optimal paths which avoid the obstacles and bypass them to reach the destination. Several algorithms have been presented to find the shortest paths in the presence of obstacles, which are modeled as rectangles, rectilinear polygons and general polygons [6]–[11]. In this paper, a new method to bypass the obstacles without touching them is presented.

Bends or corners are unavoidable in graphs which have obstacles and other physical constraints, or when a direct link fails. The presence of bends along the paths reduces velocity of moving objects and increases energy consumption. In physical implementation, the cost of such paths is higher as well. Therefore, the author's aim is to ensure a minimum number of bends.

Let us consider a scenario where several shortest paths of equal length exist between a pair of nodes in a rectilinear graph. Among these shortest, equal paths, the path that has the lowest number of bends is called the rectilinear minimum bend shortest path (MBSP) [6]. First, we find the MBSP between a given source-destination pair. If more than one such MBSP is present, we take one of them. The objective is to find MBSPs between K distinct source-destination pairs. This results in K distinct MBSPs, one for each source-destination pair. Additionally, the paths should be distinct, disjoint and non-crossing. In the proposed method, each source-destination pair contributes a single optimal path. We have K such paths corresponding to K such pairs. This is not the same as the first K-shortest paths for a single source-destination pair. When obstacles are present in the graph, the paths should not touch the obstacles but should avoid and bypass them. The given region where the paths have to be determined is discretized by a uniform square grid of a suitable size. Then, the constrained paths are determined on the undirected graph using the well-known Floyd-Warshall algorithms from the graph theory.

Even though multi-criteria optimization methods [12], [13] can be used to solve this MBSP problem, they are more complex and experience difficulty in getting the best weights for combining multiple criteria into a single one. The weights depend on the size and layout of the graph. The paper provides a comparison of the proposed method with the bi-criteria optimization method.

2. Basic Symbols Notations and Assumptions

The geometric region under consideration is represented by a square mesh grid which covers the entire region as shown in Fig. 1. A planar graph $\mathbf{G}(\mathbf{V},\mathbf{E})$ is constructed

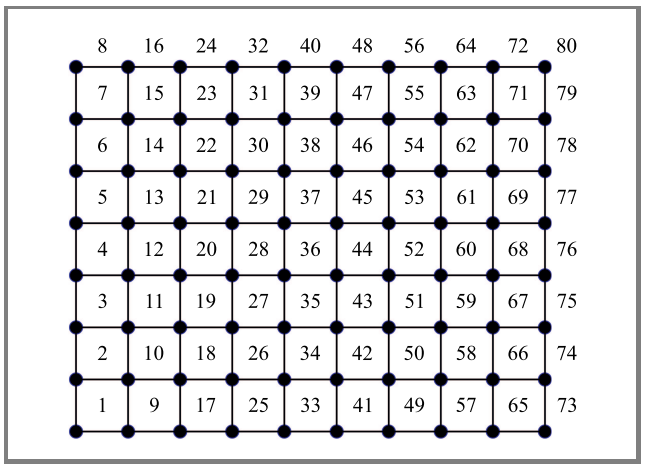

Fig. 1. Grid graph numbering scheme: width in terms of grid points $= W = 10$, height in terms of grid points $= H = 8$.

from the grid. The grid points are the vertices (nodes) of the graph and the grid lines are the edges of the graph. The number of grid points along width of the graph is W, and that along the height is H. Thus, the total number of vertices (grid points) represented by n is, $n = WH$. The total number of links would be $(W-1)H + (H-1)W$.

2.1. Node Numbering

The n nodes of the graph are numbered from 1 to n, column-wise, starting from the bottom left corner, as shown in Fig. 1. After numbering a column from its bottom to top, the next column is numbered further from its bottom to top, and so on. In each column, as we traverse from its bottom to top, the node numbers increase by one for each successive node. On the other hand, along each row, the node numbers increase by H as we traverse successive nodes from left to right. Thus, in Fig. 1, as we traverse from left to right, along any row, the node numbers increase by 8, successively. Here $H = 8$. The node numbers act as node IDs. A horizontal or a vertical line segment is denoted by the beginning and ending of that segment. Thus, for example, in Fig. 1, segments $2 \rightarrow 10$ and $80 \rightarrow 8$ are horizontal segments, whereas, $1 \rightarrow 2$ and $80 \rightarrow 73$ are vertical segments. The minimum length of a horizontal line segment, in terms of the difference between its end node IDs, is H. For example, for the line segment $2 \rightarrow 10$, the length is $10 - 2 = 8 = H$. This property applies to all horizontal segments. Since the minimum length is H, an important characteristic of horizontal segments is that they have lengths greater than or equal to H. For vertical segments, the minimum and the maximum lengths (in terms of the end node differences) are 1 and $H-1$, respectively. For example, for the minimum vertical segment $1 \rightarrow 2$, its length is $2 - 1 = 1$. For the maximum vertical segment $73 \rightarrow 80$, the length is $80 - 73 = 7 = H - 1$. Thus, for vertical segments, the maximum length is $H-1$ which means the lengths are less than H. These properties are essential to check for the existence of bends (corners) along a path, as described later. The vertex set V of the graph is $\{1 : n\}$.

2.2. Node Connectivity and Edge Weights

4-connectivity for all the nodes (vertices) is used. This means that each non-border node is connected to its 4 immediate neighbors: north, south, east and west. The four corner nodes of the graph have 2 connections each. Non-corner border nodes have 3 corners each. Thus, the graph is a one-hop network. The nodes which are more than one hop are not connected directly at all. The weights (effective distances) for the connecting edges are taken as one. In this paper, the terms weight and length are used synonymously. Thus, the length of an edge is the same as the weight of that edge. The weights for the edges between directly unconnected nodes are set to infinity. All edge weights are positive. Thus, an edge weight of ∞ means no direct connection. Therefore, the elements of the adjacency matrix are either 1 or ∞.

2.3. Adjacency Matrix

The adjacency (connectivity) matrix of the graph is represented by $\mathbf{A}$. The size of matrix $\mathbf{A}$ is $n \times n$. The element $\mathbf{A}(i,j)$ of the adjacency matrix gives the weight of the edge (link) between node i and node j. The diagonal elements of $\mathbf{A}$ are taken as ∞, to prevent self-loops. The graph is an undirected graph. Therefore, $\mathbf{A}(i,j) = \mathbf{A}(j,i)$ and $\mathbf{A}$ is symmetric. An edge can allow the flow in either direction. When i and j are connected $\mathbf{A}(i,j) = 1$, else, $\mathbf{A}(i,j) = \infty$. The number of 1s in row i of $\mathbf{A}$ gives the number of edges leaving node i, and the number of 1s in column i gives the number of edges entering node i. Since our graph is a 4-connected grid graph, the maximum number of 1s in a row or column is 4. As an example, consider a 3×3 grid graph of nine nodes represented by $\mathbf{G}$, as:

$$\mathbf{G} = \begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix}.$$

The corresponding adjacency matrix $\mathbf{A}$ will be:

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{bmatrix} \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 1 & \infty & 1 & \infty & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty & \infty & 1 & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 1 & \infty & 1 & \infty & \infty \\ \infty & 1 & \infty & 1 & \infty & 1 & \infty & 1 & \infty \\ \infty & \infty & 1 & \infty & 1 & \infty & \infty & \infty & 1 \\ \infty & \infty & \infty & 1 & \infty & \infty & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty & 1 \\ \infty & \infty & \infty & \infty & \infty & 1 & \infty & 1 & \infty \end{bmatrix} \end{array}.$$

In general, in $\mathbf{A}$, the rows, as well as the columns of the corner nodes, have 2 ones, those of the non-corner border nodes have 3 ones and those of the inside nodes have 4 ones.

Thus, the nodes are characterized by direct connectivity, immediately along north↔south and east↔west axes. The nodes have no immediate diagonal connectivity. This is an important requirement to ensure the non-crossing property of the shortest paths. Because of this 4-connectivity, all connected paths from the source node to the destination node have to be rectilinear.

2.4. Objective

Given the source-destination node pairs $(s_i - t_i)$ for $i = 1$ to K, the main objective is to find K shortest paths $\mathbf{P}(s_i,t_i)$'s which are disjoint, having a minimum number of bends and which do not touch any obstacles present in the graph.

3. Properties of Paths

A simple path from the source node s to the destination node t is a series of connected edges from s to t through the intermediate nodes without any loops. Hereafter, for brevity and convenience, we refer to simple path(s) by the term path(s). Let the sequence of intermediate nodes along a specific path from s to t be, $v_1, v_2, \ldots, v_m$. Then, the entire path including s and t is represented by $\mathbf{P}(s,t)$, such as:

$$\mathbf{P}(s,t) = [\, s, v_1, v_2, \ldots, v_m, t]\,. \tag{1}$$

$\mathbf{P}(s,t)$ is an array of nodes. With m intermediate nodes, the size of the array is $m+2$, which is same as the number of nodes in $\mathbf{P}(s,t)$. The length of the path is the sum of the edge weights along the path from s to t. From Eq. (1), one can see that the number of edges connecting s to t is $m+1$, that is one less than the size of the array $\mathbf{P}(s,t)$. In this case, the weights of all the connecting edges are 1s. Therefore, the total weight of the path is $m+1$ itself. The total weight of the path is also called the length of the path or path length. When the path length is short, then the corresponding travel cost is also lower.

3.1. Shortest path

When there are several paths from s to t, that path which has the minimum path length is the shortest path. The number of shortest paths can be more than one. Then, the path lengths of these are minimum and equal.

3.2. Rectilinear property of paths

Our graph is a square grid graph and all the edges are either parallel to x axis or y axis. Since a path is a chain of edges, the edges making up the path are parallel to the Cartesian coordinates. Then the path is rectilinear, because each edge of the path is either parallel to x or y, i.e. each edge is either vertical or horizontal.

3.3. Non-crossing Property of Paths

To understand the non-crossing property of the paths, we introduce a theorem on the non-crossing constraint.

Theorem 1. In a square grid graph, node-disjoint paths do not cross each other. *Proof.* In the presented scheme, all paths are rectilinear. Consider two paths $\mathbf{P}(s_1,t_1)$ and $\mathbf{Q}(s_2,t_2)$, where the source and the destination nodes s_1, t_1, s_2, t_2 are all disjoint. Let $E_P(a,b)$ be any edge that belongs to $\mathbf{P}(s_1,t_1)$ and similarly let $E_Q(c,d)$ be any edge that belongs to $\mathbf{Q}(s_2,t_2)$. Now, consider the geometrical intersection of edges $E_P(a,b)$ and $E_Q(c,d)$. When both of them are horizontal or both of them vertical, they are parallel

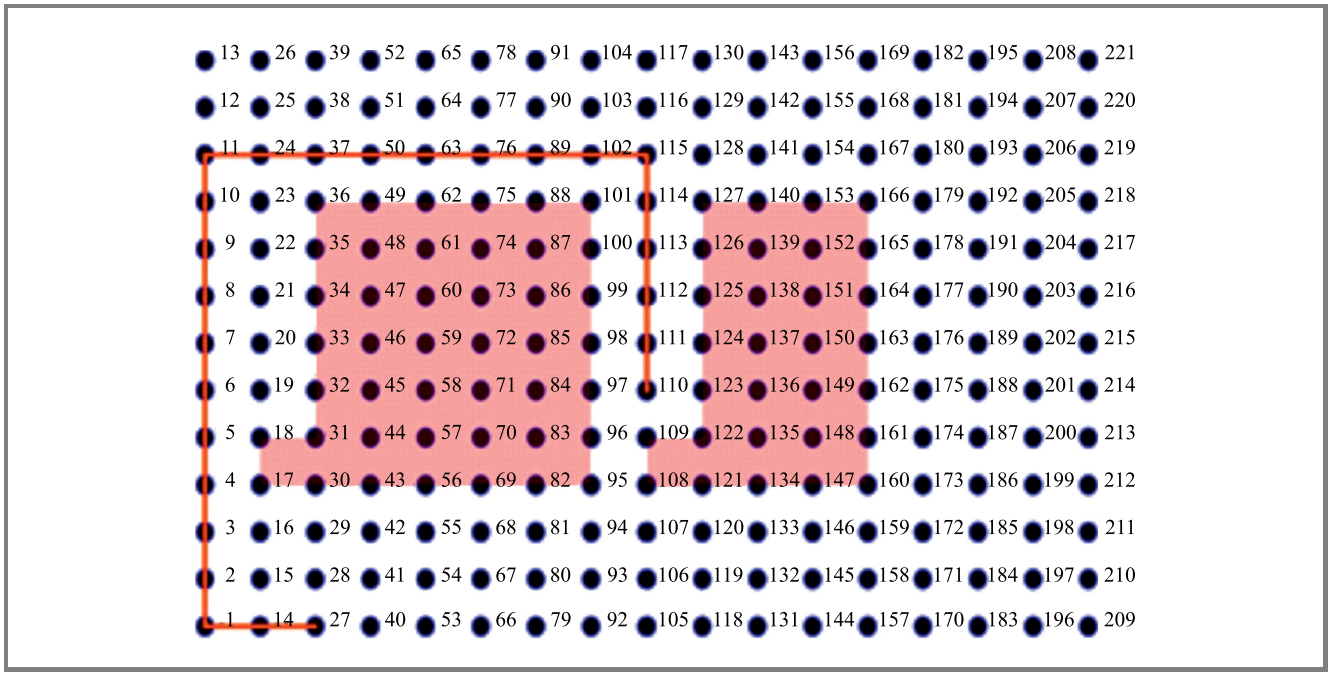

Fig. 2. Obstacles bounded by grid points marked by shaded areas.

and cannot meet each other. Therefore, the condition for the intersection is that one of them has to be horizontal while the other one vertical. In our grid graph, vertical and horizontal edges meet only at nodes (see Fig. 1). Hence, the two edges can meet only at one of the nodes. This node is obviously the common node for both the edges. Therefore, the condition for the intersection of $E_P(a,b)$ and $E_Q(c,d)$ is that they should have a common node or the paths should not be disjoint. On the other hand, if $E_P(a,b)$ and $E_Q(c,d)$ are node-disjoint, they cannot meet each other. Since the paths are made up of a series of edges, if the edges of the respective paths cannot meet, then the two paths also cannot intersect. This argument can be extended to all possible paths. Therefore, if paths are node-disjoint, they cannot intersect each other. Based on Theorem 1, determination of non-crossing paths boils down to that of finding node-disjoint paths.

3.4. Representation of Obstacles in the Graph

Obstacles are those regions in the graph which should be avoided by the paths. The paths should not touch the obstacles and have to bypass them if necessary to reach the destination. In this grid graph, the boundaries of the obstacles are marked by the grid points (nodes of the graph), as shown in Fig. 2. Since obstacle boundaries are represented by the grid points of the square grid graph, the obstacles are rectilinear polygons. The obstacle can be in the form of polylines, as shown by the line in Fig. 2. We can also specify certain grid points as obstacles. Then, all points along the line and the isolated obstacle points are excluded or made inaccessible while finding the shortest paths.

3.4.1. Exclusion (or marking out) of a specific node

Exclusion of a specific node is done by setting the weights (distances) of all incident edges of that node to infinity. Then, the edges entering or leaving that node have ∞ as their weight. Therefore, the shortest path algorithm will exclude that node, because if included, the total length (weight) would become ∞. Let the node to be excluded be J, where $J \in \{1:n\}$. Then the weights of all incident edges leaving J are given by the elements of row J of the adjacency matrix $\mathbf{A}$. Similarly, the weights of incident edges entering J are given by the elements of column J of $\mathbf{A}$. Therefore, the elements of row J and column J have to be set to ∞. Before setting the elements to ∞, matrix $\mathbf{A}$ is saved in a temporary matrix **Atemp**, which can be used to restore J as explained later. Thus, the exclusion operation is carried out as:

$$\mathbf{Atemp} = \mathbf{A}\,, \tag{2}$$

$$\mathbf{A}(J,:) = \infty\,. \tag{3}$$

Here, the colon notation is used to represent all elements of row J. Similarly the weights of incident edges entering J are set to ∞ as:

$$\mathbf{A}(:,J) = \infty\,, \tag{4}$$

Here, $\mathbf{A}(:,J)$ represents all the elements of column J. Thus, use of Eqs. (3)–(4) alters the adjacency matrix $\mathbf{A}$ such that node J is excluded or **marked out** while searching for the shortest path. Nodes belonging to the obstacles are permanently marked out. But in our method, certain nodes are **marked out** temporarily in the present iteration, to be restored in the later iterations as will be explained later.

3.4.2. Inclusion or mark-in of a node that was excluded earlier

Inclusion or **mark-in** of node J which was marked out earlier is done by restoring the elements of row J and column J from $\mathbf{A}$ as:

$$\mathbf{A}(J,:) = \mathbf{Atemp}(J,:)\,, \tag{5}$$

$$\mathbf{A}(:,J) = \mathbf{Atemp}(:,J)\,. \tag{6}$$

3.5. Node-disjoint K Shortest Paths

Determination of node-disjoint shortest paths is a well-known NP-complete problem [6]. To overcome this, Yen's iterative method [7] is used, which is sufficient for engineering applications. Given K distinct source-destination node pairs (s_i,t_i) for $i=1$ to K, we first find the shortest path from s_1 to t_1. Then, the nodes along the first shortest path are made inaccessible for the next iteration, by setting the weights of all incident edges of those nodes to infinity (**marked-out** operation), as given by Eqs. (2)–(3). Next, we find the second shortest path. Again, make the involved nodes inaccessible by setting the relevant incident edge weights to infinity (**marked-out** operation) and so on, until we determine K shortest disjoint paths. In this method, the next iteration will exclude those nodes which have been already covered by the previous paths. We use Floyd-Warshall all pairs shortest path algorithm for determining the individual shortest paths. The reason for using this algorithm is described later.

3.6. Bends in a Path

Consider a path $\mathbf{P}(i,j)$ from source i to destination j. Let k be an intermediate node along the path, as shown in Fig. 3. Symbol, k (lowercase) is used for intermediate nodes in the Floyd-Warshall algorithm. This k is different from the uppercase K which represents the number of shortest paths. Now, $\mathbf{P}(i,k)$ and $\mathbf{P}(k,j)$ are the two segments of $\mathbf{P}(i,j)$ joined at node k. The two segments can be perpendicular, as in Fig. 3a, or collinear (no bend), as in Fig. 3b.

3.6.1. Checking for a bend at *k*

For the existence of a bend in a path, one line segment should be horizontal and the other one has to be vertical and vice-versa. As explained earlier, the condition for the segment $\mathbf{P}(i,k)$ to be horizontal is, its end node difference $\text{abs}(i-k)$ should be greater than or equal to H. That is, $\text{abs}(i-k) \geq H$. The condition for segment $\mathbf{P}(k,j)$ to be

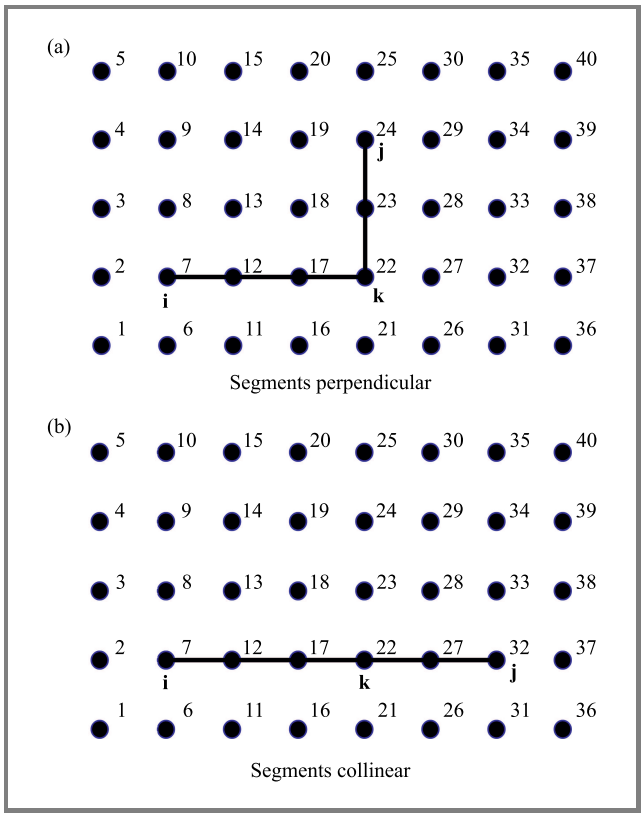

Fig. 3. Two possible paths from node i to node j via k.

vertical is, $\text{abs}(k-j) < H$. These two conditions can be expressed as:

$$\begin{cases} \text{abs}(i-k) \geq H \text{ , if } \mathbf{P}(i,k) \text{ is horizontal} \\ \text{abs}(k-j) < H \text{, if } \mathbf{P}(k,j) \text{ is vertical} \end{cases}. \quad (7)$$

Constraint (7) gives the condition for $\mathbf{P}(i,k)$ and $\mathbf{P}(k,j)$ to be perpendicular. They can also be perpendicular when $\mathbf{P}(i,k)$ is vertical and $\mathbf{P}(k,j)$ is horizontal. This condition can be expressed as:

$$\begin{cases} \text{abs}(i-k) < H \text{ , if } \mathbf{P}(i,k) \text{ is vertical} \\ \text{abs}(k-j) \geq H \text{, if } \mathbf{P}(k,j) \text{ is horizontal} \end{cases}. \quad (8)$$

If either constraint (7) or (8) holds true, then $\mathbf{P}(i,k)$ and $\mathbf{P}(k,j)$ are perpendicular. Constraints (7) and (8) are combined using OR logic and the combined logical constrain is written as:

$$G = \big(\text{abs}(i-k) \geq H \text{ \&\& } \text{abs}(k-j) < H\big) \\ || \ \big(\text{abs}(i-k) < H \text{ \&\& } \text{abs}(k-j) \geq H\big). \quad (9)$$

Here, && is the logical AND operator, || is the logical OR operator and G is a logical variable. $\mathbf{P}(i,k)$ and $\mathbf{P}(k,j)$ are perpendicular if G given by Eq. (9) is true, else they are collinear. In Fig. 3, $G =$ true for the path shown in Fig. 3a and $G =$ false for the path shown in Fig. 3b.

4. Modified Floyd-Warshall Algorithm

The main contribution of this paper is the modification of the Floyd-Warshall all pairs shortest path algorithm to take care of the 90° bends (corners) along the paths.

4.1. Basic Principle

Consider two different paths $\mathbf{P}(s,a,t)$ and $\mathbf{Q}(s,b,c,d,e,f,t)$ from source s to destination t, as shown in Fig. 4. Here, $\mathbf{P}(s,a,t)$ has one bend at a, while $\mathbf{Q}(s,b,c,d,e,f,t)$ has 5 bends at b,c,d,e,f.

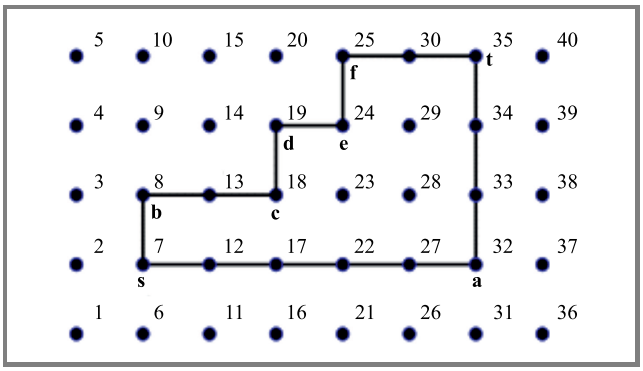

Fig. 4. Two different paths having the same length, but different number of bends.

In the square grid graph of Fig. 4, every connecting edge has a weight 1 and the paths are rectilinear. Therefore, the lengths of the two paths are the same and equal 8, even though the number of bends in each path is different. In this case, the shortest path algorithm can choose either path **P** or path **Q**, with equal preference. But our objective is to select **P** against **Q**, because of a lower number of bends in **P**. To achieve this objective, *corner weights* are introduced in addition to edge weights.

4.1.1. Corner weights

We assign a very small weight (compared to the edge weight) ε to each corner (or bend). Thus, the corner weight is represented by ε. Now, while calculating the total length of a path, both the edge weights and the corner weights are added. This total length is called the augmented length of a path. A very low corner weight is chosen, so that while comparing two paths of different total edge weights, it does not play any significant role. On the other hand, while comparing two different paths of equal total edge weight, the corner weights play a decisive role.

4.1.2. Selection of corner weight ε

Let the possible estimated maximum number of bends (corners) along a lengthy path in the graph be denoted by *mnb*. Then the total corner weight would be $mnb \cdot \varepsilon$. This value should be less than one hop weight, so that the addition of corner weight does not affect the relative weights of paths

under comparison. Therefore, ε should be chosen such that, $mnb \cdot \varepsilon < 1$. That is, choose ε as:

$$\varepsilon = \frac{1}{mnb} \tag{10}$$

Thus, ε depends on the size and layout of the graph.

4.1.3. Augmented path length

The total length of a path considering both the edge weights and corner weights is defined as the augmented path length (APL). Thus for a given path:

APL = "sum of edge weights along the path"
+ "sum of corner weights along the path".

Therefore, a rectilinear path with a lower number of corners will have a lower APL. For example, in Fig. 4, the APL of path $\mathbf{P}(s,a,t)$ is $8+\varepsilon$, because it has one corner, whereas that of path $\mathbf{Q}(s,b,c,d,e,f,t)$ is $8+5\varepsilon$, because it has 5 corners.

Algorithm 1: Floyd Warshall all pair shortest path

$[\mathbf{D}, \mathbf{Next}] = \text{FWAPSP1}(\mathbf{A}, H, n)$

1: **For** each edge (u,v)
2: $D[u][v] \leftarrow A[u][v]$; // $A[u][v]$ is same as $A(u,v)$.
3: $Next[u][v] \leftarrow v$;
4: **End for**
5: **For** $k = 1$ to n
6: **For** $i = 1$ to n
7: **For** $j = 1$ to n
8: **If** $D[i][j] > D[i][k] + D[k][j]$
// update for minimization
a: $D[i][j] \leftarrow D[i][k] + D[k][j]$;
b: $Next[i][j] \leftarrow Next[i][k]$;
// Modification to check for a 90° bend
c: $G = (\text{abs}(i-k) >= H \text{ \&\& } \text{abs}(k-j) < H)$
$||(\text{abs}(i-k) < H \text{ \&\& } \text{abs}(k-j) >= H$;
If (G==true)
$D[i][j] \leftarrow D[i][j] + \varepsilon$;
End if
// Modification end
9: **End if**
10: **End for** // j
11: **End for** // i
12: **End for** // k

Since APLs take care of both the edge weights and corner weights, we use APLs instead of just edge weights, in the shortest path algorithm. Then the algorithm determines the shortest paths with minimum number of bends.

4.2. Floyd-Warshall Algorithm with Corner Weights

The modified Floyd-Warshall algorithm is given as Algorithm 1. On input, it takes adjacency matrix **A**, graph height **H**, number of nodes n and corner weight value ε. On output, it gives the shortest path length (distance) **D** matrix and **Next** matrix. Algorithm 1 is named as Floyd Warshall all pair shortest path (FWAPSP1), and it is written in the form of a function.

Algorithm 2: Function FWAPSP2

$[\mathbf{F}, \mathbf{Next}] = \text{FWAPSP2}(\mathbf{D}, H, n)$

1: **For** each edge (u,v)
2: $F[u][v] \leftarrow D[u][v]$; // $[u][v]$
3: $Next[u][v] \leftarrow v$;
4: **End for**
5: **For** $k = 1$ to n
6: **For** $i = 1$ to n
7: **For** $j = 1$ to n
8: **If** $F[i][j] > F[i][k] + F[k][j]$
// update for minimization
a: $F[i][j] \leftarrow F[i][k] + F[k][j]$;
b: $Next[i][j] \leftarrow Next[i][k]$;
// Modification to check for a 90° bend
c: $G = (\text{abs}(i-k) >= H \text{ \&\& } \text{abs}(k-j) < H)$
$||(\text{abs}(i-k) < H \text{ \&\& } \text{abs}(k-j) >= H$;
d: **If** (G==true)
$F[i][j] \leftarrow F[i][j] + \varepsilon$;
End if
// Modification end
9: **End if**
10: **End for** // j
11: **End for** // i
12: **End for** // k

D matrix gives the minimum APL connected paths between all pairs of nodes. In Algorithm 1, we see that the update of $D[i][j]$ and $Next[i][j]$ (steps 8a and 8b) occur before the augmentation of $D[i][j]$ by the corner weight ε. Therefore, matrix **D** and **Next** will not fully and truly reflect the overall connectivity of the paths taking the APLs into account. Hence, we have to recalculate new **D** and **Next**, based on $D[i][j]$'s instead of $A[i][j]$'s. Even though, the **Next** matrix calculated from FWAPSP1 is not directly used further, it is retained as a formality. The recalculation of matrices **D** and **Next** is implemented in Algorithm 2. On input, it takes

connectivity matrix **D**, graph height H and n. On output, it gives updated shortest path length (distance) **F** matrix and **Next** matrix.

Checking for a bend in the path at k, an intermediate node between i and j, is straightforward in the Floyd-Warshall algorithm compared to Dijkstra, because both the terms on the RHS of step 8a (update for minimization) of FWAPSP1 or FWAPSP2 are an addition of two sub-paths $\mathbf{P}(i,k)$ and $\mathbf{P}(k,j)$. Therefore, the constraint term G, as given by Eq. (9) can be easily constructed, but, in Dijkstra, it is not the case.

4.3. Main Algorithm

The objective is to determine K number of rectilinear non-crossing shortest minimum bend paths (RNMBP) in the presence of rectilinear obstacles. The main algorithm uses Algorithm 1 and 2 to implement RNMBP. Initially, the given obstacles are **marked out** in the adjacency matrix **A**.

Algorithm 3: RNMBP

1: Get the matrix **A** for the given graph with obstacles marked out, use Eqs. (2) and (3)
2: Find the K shortest paths for the given (s,t) ignoring the non-crossing property using unmodified Floyd-Warshall algorithm
3: Get the sorted (s,t) pair based on the lengths of the shortest paths obtained in step 2
4: **Atemp** $=$ **A** // store **A** for the future use
5: Set $i = 1$ // start with the first pair
6: Mark out (s,t) nodes for indices greater than i as obstacles. // we do not want the present path to pass // through those higher indices (s,t) nodes
7: Get **D** using, $[\mathbf{D},\mathbf{Next}] = \text{FWAPSP1}(\mathbf{A},H,n)$. // the return value **Next** is not used
8: Get **Next** using, $[\mathbf{E},\mathbf{Next}] = \text{FWAPSP1}(\mathbf{D},H,n)$
9: **If** $\mathbf{E}(s_i,t_i) = \infty$ go to step 12 // this path not exists
10: Reconstruct the path $\mathbf{P}(s_i,t_i)$
11: Mark out all the nodes along this path.
12: Mark in (restore) the marked out (s,t) node pairs using **Atemp**
13: Increment i for the next iteration as $i = i+1$
14: **If** $i > K$ exit, **Else** go to step 6
15: Exit

4.3.1. Order of processing

In a graph, a longer path creates more obstacles for the succeeding paths than a shorter path. Consider the example of two paths shown in Fig. 5.

The longer line is between 3–38 nodes and the shorter one is between 24–22. If the longer line is drawn first as in Fig. 5a, it covers the entire graph horizontally. Therefore, the second shorter line cannot be drawn without crossing the first line. Hence the non-crossing property cannot be satisfied. On the other hand, if the shorter line is first, the obstacle created is small and the second longer line can be drawn satisfying the non-crossing property, as shown in Fig. 5b.

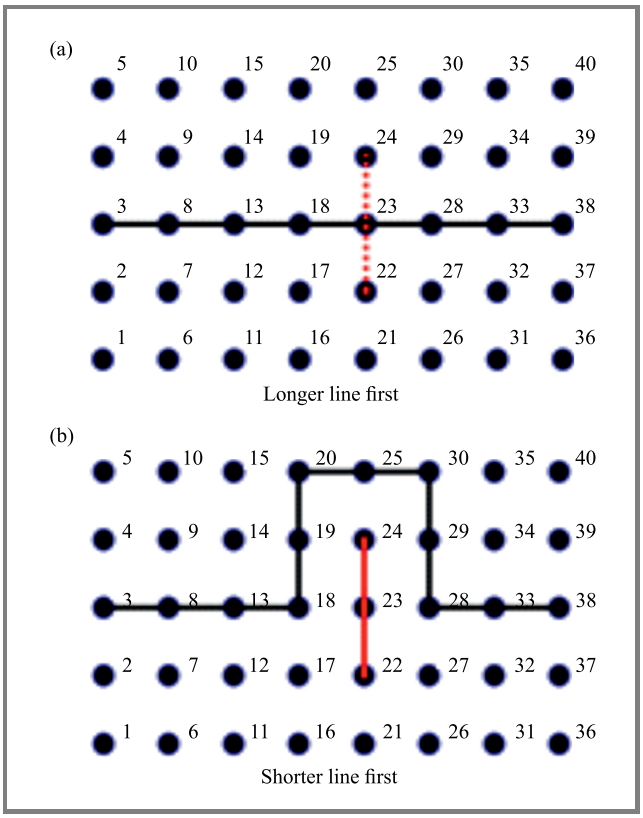

Fig. 5. The effect of order of preference.

Therefore, among the K shortest paths to be determined, we process them in the increasing order of their lengths. Initially, the shortest lengths of all K paths are determined considering the non-crossing (node disjoint) property. Then the corresponding (s,t) pairs are sorted in the increasing order of their lengths. Then, the sorted (s,t) list is processed one after another, as described in Subsection 3.5.

Now, $\mathbf{P}(s_i,t_i)$ for $i = 1$ to K gives the K shortest non-crossing minimum bend paths. Since algorithm RNMB uses the modified Floyd-Warshall algorithm, the time complexity is $O(n^3)$. The path reconstruction function has a complexity of $O(n)$.

4.4. Experimental Results

Several examples are solved using RNMBP. Because of the limited space, all results are not presented.

Example 1. Here, $W = 17$, $H = 13$ and $K = 7$. The total number of nodes $n = 17 \times 13 = 221$ and $\varepsilon = 0.001$,

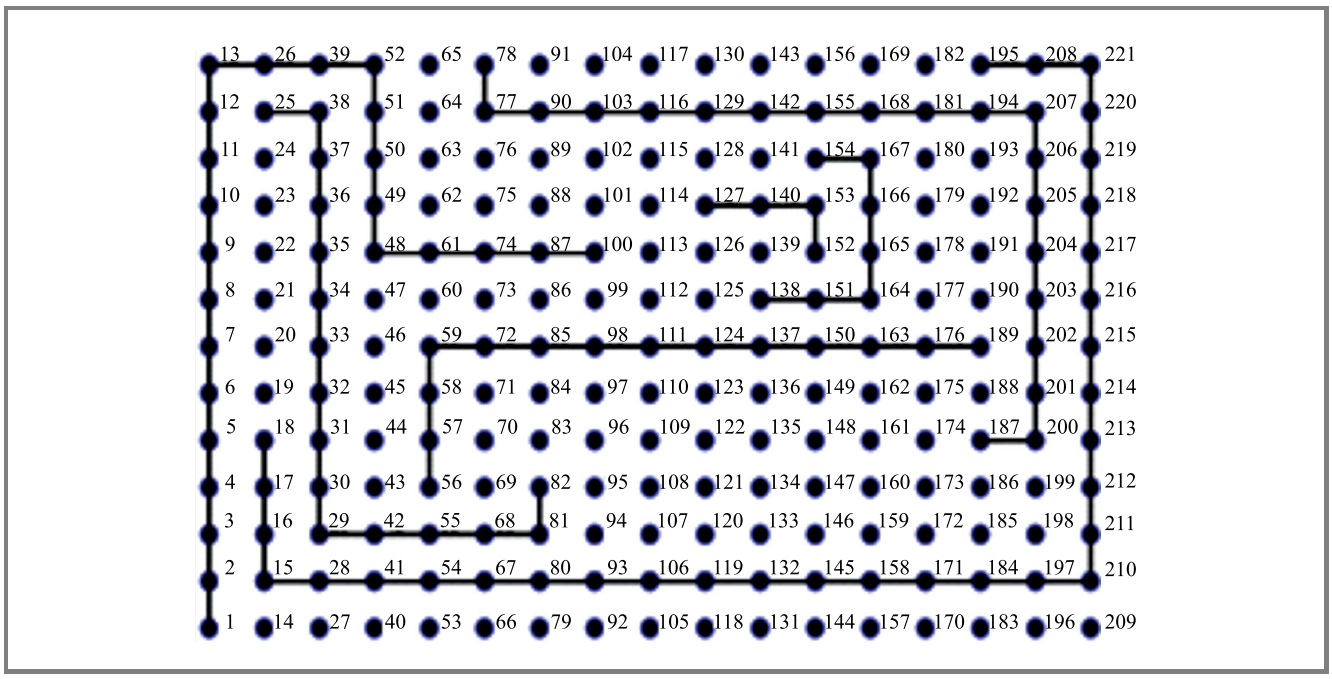

Fig. 6. Seven non-crossing shortest paths identified by RNMBP.

which corresponds to $mnb = 1000$. The $(s-t)$ pair set is taken as:

$$(s-t) = \{\ (78-187), (189-56), (127-152), (195-18), (82-25), (154-138), (1-100)\ \}$$

This example has no obstacles.
The shortest paths are found using RNMBP. The 7 non-crossing paths are shown in Fig. 6. The full path and the lengths are shown in Table 1.

Example 2. This is an example with rectilinear obstacles. Here, $W = 17$, $H = 13$ and $K = 2$. The total number of nodes $n = 17 \times 13 = 221$ and $\varepsilon = 0.001$. The $(s-t)$ pair set is taken as:

$$(s-t) = \{(1-73), (164-196)\}.$$

The two shortest paths avoiding the obstacles are found using RNMBP and are shown in Fig. 7.

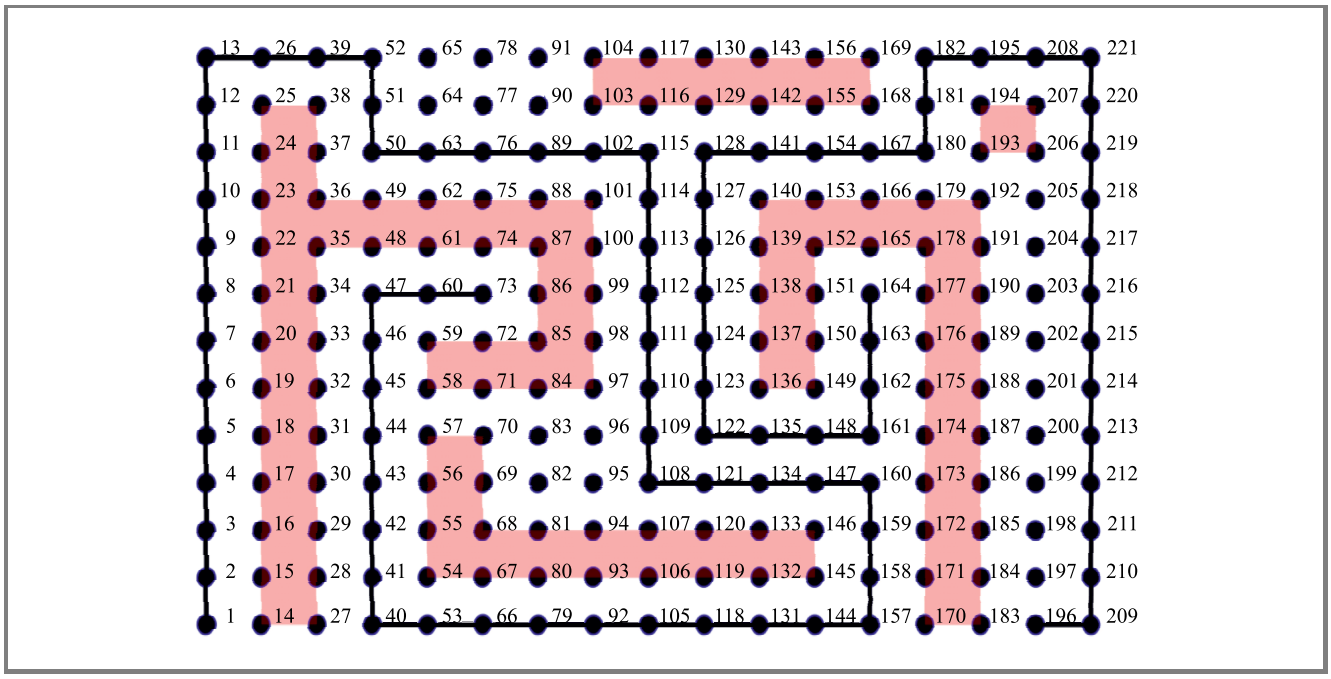

Fig. 7. Two non-crossing shortest paths avoiding obstacles identified by RNMBP.

Table 1
Nodes along the path and path lengths

i	$(s_i - t_i)$	Nodes along the full path	Path length
1	127–152	[127 140 153 152]	3
2	154–138	[154 167 166 165 164 151 138]	7
3	189–56	[189 176 163 150 137 124 111 98 85 72 59 58 57 56]	13
4	82–25	[82 81 81 68 55 42 29 30 31 32 33 34 35 36 37 38 38 25]	17
5	1–100	[1 2 3 4 5 6 7 8 9 10 11 12 13 13 26 39 52 51 50 49 48 48 61 74 87 100]	25
6	78–187	[78 77 77 90 103 103 116 129 142 155 168 181 194 207 206 205 204 203 202 201 200 200 187]	22
7	195–18	[195 208 221 220 219 218 217 216 215 214 213 212 211 210 210 197 184 171 158 145 132 119 106 93 80 80 67 54 41 28 15 16 17 18]	33
The total length of all paths = 120			

In Fig. 7, the paths do not touch the obstacles. The RNMBP algorithm can be modified so that the paths may touch the obstacle boundaries but should not cross them.

5. Comparison with Other Methods

In the proposed RNMBP algorithm, minimization of the number of bends, as well as the length of a path, is done simultaneously using the modified, well-known Floyd-Warshall algorithm, whereas in [6], the authors determine the shortest staircase-path first and then use the push-and-drag method to reduce the number of bends. The algorithm given by [6] is relatively more complex than the presented method wherein pushing and dragging are not required.

In the proposed algorithm, the obstacles are taken care of by simply marking out (excluding) the corresponding boundary nodes. In [6], an extra graph called the boundary graph is created, which unnecessarily increases complexity. In [7], track graphs are created to take care of obstacles. In [8], induced points and multiple paths are generated before getting the final path. In [9], extreme points and edges of the obstacles are determined first. In this case, additional processing of the obstacles is not required. In Lee's algorithm [10], wave-front propagation is used throughout the graph until the target node is reached which is a tedious process. In [11], a visibility graph is created to take care of the obstacles. There is no need to create additional graphs here.

In the next step, a time complexity comparison with bi-objective optimization is researched. Let us consider an example, where $K = 7$ and $\varepsilon = 0.001$. The width $W = 10$ and height H is varied from 20 to 60 in steps of 5. Therefore, $n = W \times H$ varies from 200 to 500 in steps of 50. The problem is solved using the bi-objective optimization (BiOO) method [13], as well as RNMBP. The length of the path and the number of bends along the path are expressed by functions $L(\mathbf{x})$ and $B(\mathbf{x})$, respectively, where $\mathbf{x}$ is the binary decision vector [13]. The scalarized objective function $F(\mathbf{x})$ is taken as:

$$F(\mathbf{x}) = \lambda \cdot L(\mathbf{x}) + (1 - \lambda) \cdot B(\mathbf{x}), \tag{11}$$

λ is set at 0.5 to give equal weightage to both criteria. The BiOO (minimization of $F(\mathbf{x})$) is carried out using binary integer programming. The time taken by BiOO, as well as RNMBP, for different values of n (total number of nodes in the graph) is shown in Fig. 8.

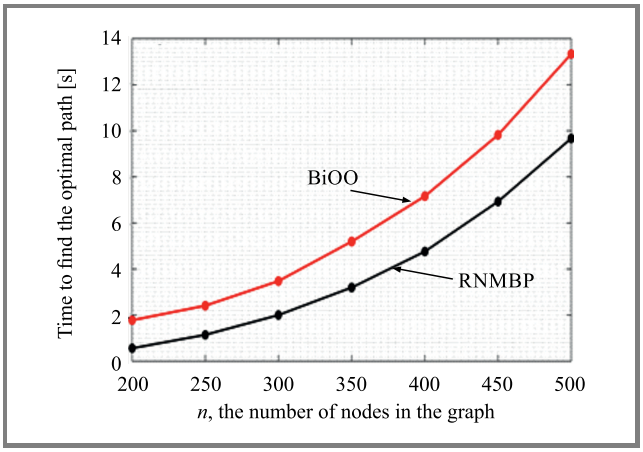

Fig. 8. Time taken versus the number of nodes.

From Fig. 8, we see that the RNMBP method takes substantially less time compared to the BiOO method. Of course, the times taken are machine-dependent and are relative only.

6. Conclusions

A new algorithm is presented to determine K shortest non-crossing minimum bend paths in the presence of obstacles, in a square grid graph. A simple and novel technique is adopted to minimize the number of bends in the shortest path by introducing corner weights and augmented path lengths in the modified Floyd-Warshall algorithm. Obstacles and non-crossing requirements are handled by **marking out** the respective nodes. Our RNMBP algorithm has a time complexity of $O(n^3)$ and is simple to implement. Even then, its task completion time is substantially lower than that of the BiOO method.

References

[1] A. Madkour, W. G. Aref, F. U. Rehman, M. Abdur Rahman, and S. Basalamah, "A Survey of Shortest-Path Algorithms" [Online]. Available: https://arxiv.org/abs/1705.02044 (accessed 6 July 2017).

[2] S. Eriksson-Bique *et al.*, "Geometric *k*-shortest paths", in *Proc. 26th Ann. ACM-SIAM Symp. Discrete Algorithms SODA'15*, San Diego, CA, USA, 2015, pp. 1616–1625 (doi: 10.1137/1.9781611973730.107).

[3] J. S. B. Mitchell, "Geometric shortest paths and network optimization", in *Handbook of Computational Geometry* J.-R. Sack and J. Urrutia (ed.). Amsterdam: Elsevier Science B.V., 1998, pp. 633–701 (doi: 10.1016/B978-044482537-7/50016-4).

[4] N. S. P. Hyun, P. A. Vela, and E. I. Verriest, "A New framework for optimal path planning of rectangular robots using a weighted L_p norm", *IEEE Robotics and Automation Let.*, vol. 2, no. 3, pp. 1460–1465, 2017 (doi: 10.1109/LRA.2017.2673858).

[5] N. Qingbin, "Research on robot obstacle avoidance and path tracking under dynamically unknown environment", in *Proc. 2017 IEEE 2nd Adv. Inform. Technol., Electronic and Automation Control Conf. IAEAC*, Chongqing, China, 2017, pp. 2607–2610 (doi: 10.1109/IAEAC.2017.8054496).

[6] C. D. Yang, O. Z. Lee, and C. K. Wong, "On bends and lengths of rectilinear paths: A graph-theoretic approach", *Internat. J. Comput. Geom. Appl.*, vol. , no. 1, pp. 61–74, 1992 (doi: 10.1142/S0218195992000056).

[7] Y. E. Wu, E. Widmayer, M. D. E. Schlag, and C. K. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles", *IEEE Trans. Comput.*, vol. C-36, no. 3, pp. 321–33, 1987 (doi: 10.1109/TC.1987.1676904).

[8] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars, "Shortest Path Queries in Rectilinear Worlds", *Int. J. of Computat. Geom. & Applic.*, vol. 2, no. 3, pp. 287–309, 1992 (doi: 10.1142/S0218195992000172).

[9] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers", *Networks*, vol. 14, no. 3, pp. 393–410, 1984 (doi: 10.1002/net.3230140304).

[10] E. M. Arkin *et al.*, "The shortest path to a segment and quickest visibility queries", in *Proc. 31st Int. Symp. on Computat. Geometry SoCG*, Eindhoven, The Netherlands, 2015, pp. 658–673 (doi: 10.4230/LIPIcs.SOCG.2015.658).

[11] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time", in *Proc. 3rd Annual ACM Sympos. Comput. Geom.*, Waterloo, ON, Canada, 1987, pp. 251–257 (doi: 10.1145/41958.41985).

[12] Z. Tarapata, "Selected multicriteria shortest path problems: An analysis of complexity, models and adaptation of standard algorithms", *Int. J. of Applied Mathem. and Computer Science*, vol. 17, no. 1, pp. 269–287, 2007 (doi: 10.2478/v10006-007-0023-2).

[13] Y. Sun and M. X. Lang, "Bi-objective optimization for multi-modal transportation routing planning problem based on Pareto optimality", *J. Ind. Eng. Manag.* vol. 8, no. 4, pp. 1195–1217, 2015 (doi: 10.3926/jiem.1562).

Shylashree Nagaraja received her B.Eng. degree in Electronics and Communication from VTU, Karnataka, India in 2006. She received her M.Tech. degree in VLSI Design and Embedded Systems from VTU, Karnataka, India, in 2008. She received her Ph.D. degree from VTU, Karnataka, India in 2016. She received Best PhD Thesis Award for the year 2016–2017 in Electronics & Communication Engineering form BITES. She is also an Associate Professor at the ECE Department, Rastreeya Vidyalaya College of Engineering, Bengaluru, Karnataka, India. She is an author or co-author of more than twenty papers on the subjects of cryptography, computer networks and VLSI.
E-mail: shylashashi@gmail.com, shylashreen@rvce.edu.in
Department of Electronics & Communication Engineering
Rastreeya Vidyalaya College of Engineering
Mysuru Road
Bengaluru 560059
Karnataka, India