INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2019, VOL. 65, NO.2 , PP. 267-276
Manuscript received September 28, 2018; revised April, 2019. DOI: 10.24425/ijet.2019.126310

Minimum Bend Shortest Rectilinear Route Discovery for a Moving Sink in a Grid Based Wireless Sensor Network

Sanu Thomas, and Thomaskutty Mathew

***Abstract*—In a rectilinear route, a moving sink is restricted to travel either horizontally or vertically along the connecting edges. We present a new algorithm that finds the shortest round trip rectilinear route covering the specified nodes in a grid based Wireless Sensor Network. The proposed algorithm determines the shortest round trip travelling salesman path in a two-dimensional grid graph. A special additional feature of the new path discovery technique is that it selects that path which has the least number of corners (bends) when more than one equal length shortest round trip paths are available. This feature makes the path more suitable for moving objects like Robots, drones and other types of vehicles which carry the moving sink. In the prosed scheme, the grid points are the vertices of the graph and the lines joining the grid points are the edges of the graph. The optimal edge set that forms the target path is determined using the binary integer programming.**

***Keywords*—Minimum bend Shortest Paths, Travelling salesman problem, Binary integer programming, Edge orientation index, Vertex Bend Index, Vertex-Edge Incident Matrix**

I. Introduction

A TRADITIONAL Wireless Sensor Network uses static sensor nodes and a static sink that collects data from the sensors over multi-hop transmission. But, when the sensors are sparsely distributed in a large geographic area, the network may not be fully connected because of the limited communication range of the sensor nodes. Then the sensors far away from the static sink may not be able to send their data to the sink. In such a scenario, a mobile sink is used to collect data from the sensors [1-5] by physically moving around the WSN. Here, it is assumed that the geographical region occupied by the sensors is suitable for the physical travel of the mobile sink and it can physically approach the area formed by the communication range of the individual sensors. In general the mobility and the scheduling of a mobile sink are deterministically controlled. The mobile sink provides a higher degree of flexibility for the efficient functioning of the WSN.

A. Moving Sink Closed Path

In general, the Mobile Sink (MS) starts from a home station, say, Base Station (BS), travels around the WSN visiting the sensors nodes, collects data and returns back to the home station. This forms a round trip tour. This process is repeated periodically depending on the nature of the application. The travel of the MS that visits different sensor nodes is similar to that of a travelling Salesman visiting the specified cities. Therefore the round trip path of the MS should be the shortest one that covers all the specified sensors.
Thus the determination of the Moving Sink Closed Path (MSCP) is same as solving the Travelling Salesman Problem (TSP). The TSP applied to the MS is called the Moving Sink Problem (MSP).

B. Rectilinear Route

In this paper, we consider a grid based WSN where the sensor nodes are placed at selected grid points. The main constraint for the MS is that it has to travel along the horizontal and vertical grid lines only and thus the MS path is a rectilinear route. More detailed descriptions are given later.

C. Minimum bend paths

Bends or corners are unavoidable along the travel path due to the topological constraints. The presence of bends or corners along the path reduces the velocity of MS and increases the energy consumed by the MS. Therefore, the number of bends has to be minimized for efficient travel of the MS.

D. Objective and methodology

The objective is to solve the MSP with minimum number of bends along the rectilinear path as the additional constraint. We use the **Binary Integer Programming** to solve this bi-objective optimization problem.

E. Organization of the paper

Section II gives a brief discussion about the related work by other authors. Section III describes the system model and the deployment of sensor nodes on the grid graph. Section IV contains the details about the MS path selection problem. Section V describes the associated constraints and their algebraic formats. Section VI formulates the binary integer program to solve the MS optimal path problem. Section VII gives the simulation results. Section VIII gives comparison with other methods. Section IX contains the conclusion.

II. Related work

V. G. Deineko , B. Klinz , A. Tiskin , G. J. Woeginger [6] have solved the Travelling salesman problems by dynamic programming. It is a modified and improved version of exhaustive search. It's time complexity is approximately

Sanu Thomas is with Faculty of School of Technology and Applied Sciences, Mahathma Gandhi University, Kottayam, Kerala, India. (e-mail: thomas.sanu@gmail.com).
Thomaskutty Mathew is with Faculty of School of Technology and Applied Sciences, Mahathma Gandhi University, Kottayam, Kerala, India. (e-mail: drtkmathew@gmail.com)

factorial. Here, the computational speed is increased using dynamic programming.

A. Maheshwari, J. R. Sack, and H. N. Djidjev [7] have provided a comprehensive survey on minimum bend paths. They have described several sequential and parallel methods for determining the shortest minimum bend paths.

C. D. Yang, O. Z. Lee and C. K. Wong [8] have used path-preserving graphs, obtained the shortest path in the staircase form and then they manipulated it by pushing and dragging to reduce the bends along that path. The method presented in [8] is a multi-stage process and relatively a slow method than our proposed method. In our method successive push and drag manipulations are absent.

K. L. Clarkson, S. Kapoor, and P. M. Vaidya, [9] create an extended graph called visibility graph and use it to determine the shortest path. But this method does not take care of minimum bend criterion.

S. Basagni, A. Carosi, C. Petrioli [10] have used Mixed Integer Linear Programming (MILP) to solve for the shortest path tour for the mobile sink in a WSN with conservation of energy as the main goal. Here, the minimization of the number of bends along the path is not discussed.

D. P. Wagner, R. S. Drysdale, C. Stein [11] have described determination of the minimum bend path using successive search method for getting 0-bend path, 1-bend path, 2-bends path and so on. The method is similar to the exhaustive search method and the time taken to get the final result is relatively very high.

M. Diaby [12] has solved the TSP in polynomial time. Basically he uses network flow linear model to solve TSP. He has also included a non-linear programming method to solve the TSP.

G. Pataki [13] has used linear integer programming to solve the TSP. He has applied branch and bound technique to arrive at the solution.

Several works have been carried out on different versions of TSP and minimum bend paths. But the combination of these two has not been discussed by any author. Therefore, the minimum bend shortest rectilinear path discovery method for the moving sink is an innovative one in WSN.

III. System model

In a WSN, the sensor nodes can be deployed randomly or deterministically. In random deployment, some areas may get over populated and other regions may receive less number of sensor nodes causing unequal coverage. Also, the location of sensor nodes is not under the control of the network designer. Therefore, in the proposed scheme, we adopt grid based deployment where the locations of the sensors can be fixed accurately and the network topology can be designed appropriately to suit the present application. Grid based WSN's are efficiently replicable and scalable.

In our scheme, the WSN layout is modeled as a uniform grid graph as shown in Fig. 1. The grid cells are squares of the same size throughout. Horizontal and vertical edges are displayed in green and magenta respectively. The sensor nodes are sparsely deployed at specified grid points and shown in blue. All grid points need not be occupied by the sensor nodes. The unoccupied grid points can be used for future expansion.

The size of the layout is taken as $W\times H$ where W is the width and H is the height expressed in terms of the grid points. Thus the total number of grid points is $(W\times H)$. A planar undirected graph $\boldsymbol{G(V, E)}$ is formed by this grid. The grid points are the vertices of the graph and the grid lines are the edges of the graph. The total number of grid points (vertices) in the grid graph represented by N is N= $W*H = |\boldsymbol{V}|$. From the grid graph layout, we can see that the number of vertical edges between adjacent nodes is $(H-1)*W$ and similarly, the number of horizontal edges is $(W-1)*H$. Thus the total number of edges between the adjacent vertices in the graph represented by M is $M = (W-1)*H+(H-1)*W = |\boldsymbol{E}|$. The edge set of the graph is denoted as,

$$\boldsymbol{E} = [e(1), e(2),\dots, e(j), \dots, e(M)]$$

For convenient representation and usage, the j_{th} edge $e(j)$ is given the identification number j itself. The enumerating order could be any suitable one. But once numbered, the edges should stick to them consistently throughout. Thus E is given by,

$$\boldsymbol{E} = [1, 2,\dots, j, \ \dots, M] \tag{1}$$

A. Vertex numbering

For the purpose of description, the N vertices of the graph are numbered from 1 to N, column-wise. The bottom left corner is the starting point and the top right corner is the ending point as shown in Fig. 1. The vertices of the graph are identified by these numbers. The vertex set of the graph is $\boldsymbol{V} = \{1{:}\ N\}$. For convenience, the i^{th} vertex is designated as $v(i)$. Here, the id of $v(i)$ is i itself.

A special property of this numbering scheme is that, for a vertical edge, the ids of its end points differ exactly by 1, whereas the ids of end points of any horizontal edge differ exactly by H. Let an edge $e(j)$ be represented by its two end points $ep(1)$ and $ep(2)$ as.

$$e(j) = \{ep(1)\rightarrow ep(2)\} \tag{2}$$

Then we can determine the orientation of that edge by examining the difference | ep(1)−ep(2)| as,

$$e(j) \text{ is } \begin{cases} \text{vertical} & \text{if } |ep(1) - ep(2)| = 1 \\ \text{horizontal} & \text{if } |ep(1) - ep(2)| = H \end{cases} \tag{3}$$

This property will be very useful as explained later. For example, in Fig.1, for the edge (3→11), the difference is (11−3) = 8 = H. Therefore (3→11) is horizontal. On the other hand consider the edge (68→67). The difference is 1 and therefore, the edge is vertical.

B. Vertex connectivity

Here, we use 4-connectivity for all the vertices. That means a non-border vertex is connected to its immediate 4 neighbors, one along north, next one along east, another one along south and last one along west. Therefore, a non-border vertex has 4 incident edges. The degree of a vertex is the number edges incident on that vertex. Hence, the degree of a non-border vertex is four. A boundary-corner vertex has two neighbors whereas a non-corner border vertex has three neighbors. For example, in Fig. 1, vertex 38 is connected 39 (north), 46 (east), 37(south) and 29 (west). But no direct connectivity exists between vertex 38 and 40 because, they are non-adjacent. The lengths of all the edges connecting the adjacent vertices are normalized and set to 1. Thus our graph is an orthogonal graph. The length between the non-adjacent vertices is set to ∞. The vertex-edge connectivity information is represented by the vertex-edge incidence matrix. This matrix can be obtained from the adjacency matrix.

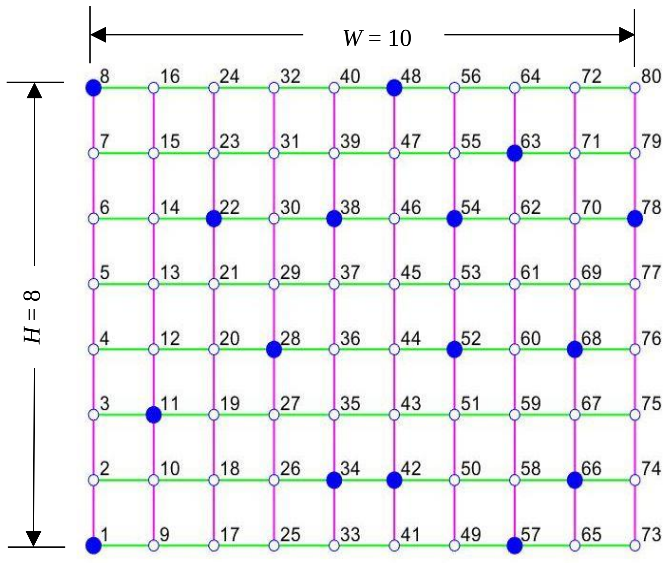

Fig. 1. WSN as a Grid graph. Width = W = 10 grid points
Height = H = 8 grid points

C. *Vertex-Edge Incidence Matrix*

The Vertex-Edge (***VE***) incidence matrix of size NxM gives the information about the end point vertices of each edge. Matrix ***VE*** has N rows that correspond to the vertices of the graph and M columns that correspond to the edges of the graph. The element of ***VE*** at row i and column j is set as,

$$ve(i,j) = \begin{cases} 1, & \text{if edge } j \text{ is incident on vertex } i \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

for i =1 to N and j = 1 to M.

D. *Properties of **VE***

1) ***VE*** is a binary matrix of size NxM.

In the grid graph of Fig.1, each edge is incident on exactly two specific vertices which are the end points of that edge. Therefore,

2) The number of ones in every column of ***VE*** is always 2.

3) From definition (4), we see that the index locations of 1's in row i represent the edges incident on vertex i.

4) Row sum, $\sum_{j=1}^{M} ve(i,j)$ gives the total number of edges incident on vertex i. This total number of edges incident on vertex i is the degree of vertex i. Therefore, the degree of vertex i for for i = 1 to N is given by,

$$\text{Degree}(i) = \sum_{j=1}^{M} ve(i,j) \quad (5)$$

Example 1: A simple graph having 9 vertices and 12 edges is shown in Fig. 2. The edge numbers are shown inside the square brackets. The ***VE*** matrix for this graph is shown in Table I.

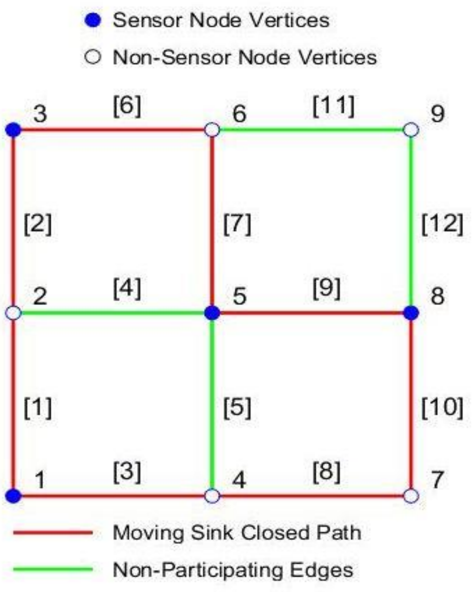

Fig. 2. Grid graph with 9 vertices and 12 edges

From Table I, It can be seen that the edges of vertex v(4) are given by the locations of 1's in row 4. They are [3, 5, 8]. The degree of v(4) = degree(4) = sum of row 4 is 3 as can be verified in Fig. 2. For vertex v(5), degree(5) = sum of row 5 is 4 which can be seen in Fig. 2. In this way, we can calculate the degree of every vertex from Table I and it can be verified in Fig. 2.

TABLE I
VE MATRIX FOR EXAMPLE 1

	EDGES →												
		1	2	3	4	5	6	7	8	9	10	11	12
VERTICES ↓	1	1	0	1	0	0	0	0	0	0	0	0	0
	2	1	1	0	1	0	0	0	0	0	0	0	0
	3	0	1	0	0	0	1	0	0	0	0	0	0
	4	0	0	1	0	1	0	0	1	0	0	0	0
	5	0	0	0	1	1	0	1	0	1	0	0	0
	6	0	0	0	0	0	1	1	0	0	0	1	0
	7	0	0	0	0	0	0	0	1	0	1	0	0
	8	0	0	0	0	0	0	0	0	1	1	0	1
	9	0	0	0	0	0	0	0	0	0	0	1	1

E. *Participating Edges and Participating Degree of a vertex*

Assume that the MS path is formed. All the edges of a vertex may not be included in the MS path. Those edges of vertex $v(i)$ that lie on the MS path are referred as the *participating edges*. If a vertex does not lie along the MS path, all the incident edges of that vertex will not participate in forming the path and therefore, the number of participating edges of that vertex is zero, The set formed by the participating edges of v(i) is represented by the symbol ***PE***(i). Parameter i refers to the vertex v(i) on which these edges incident, Thus, set ***PE***(i) gives the ids of the participating edges incident on v(i).

The count of participating edges of v(i) is denoted as the *Participating Degree PD*(i) of v(i). Thus *PD*(i) = |***PE***(i)|. For example, in Fig. 2, the full edge set of v(5) is [4, 5, 7, 9] and the full degree of v(5) is 4. Let the MS path pass through v(5) as shown in red. Then, the participating edge set of v(5) is, ***PE***(5) =[7, 9] and the participating degree is, *PD*(5) = 2.

On the other hand, non-participating edges of v(5) are [4, 5]. In general, for a given vertex *v*(i), the *PD*(*i*) is less than or equal to its full degree.

F. *Admission Control Variable for edjes*

In solving the MSP problem, the admissibility of edge *j* as a participating edge is decided by the binary admission control variable x(*j*) associated with edge *j*, for *j* = 1 to M. The admissibility condition is given by,

$$\text{edge } j \text{ is } \begin{cases} \text{admissible} \ \text{ if } x(j) = 1 \\ \text{blocked} \ \text{ if } x(j) = 0 \end{cases} \quad (6)$$

Thus x(*j*) is the binary decision variable associated with edge *j*. The sequence of all *x*(*i*)'s forms the Admission Control Vector ***X*** as,

$$\boldsymbol{X} = [x(1), x(2), \ldots, x(j), \ldots, x(\mathrm{M})] \quad (7)$$

1) *Participating Degree Calculation using the VE matrix*

From (4), we know that *ve*(*i*, *j*) = 1 gives the incident edge *j* on vertex *i*. To indicate its participation in the MS path formation, we use the product *ve*(*i*, *j*)*x(*j*). Now from (4) and (6) we see that, both *ve*(*i*, *j*) and x(*j*) have to be 1 for e(j) to be a member of the MS path. Then the corresponding *Participating Incident Edge Count* (*PIEC*) can be expressed as,

$$PIEC(i,j) = \begin{cases} 1 \ \text{ if } ve(i,j) * x(j) = 1 \\ 0, \quad \text{otherwise} \end{cases} \quad (8)$$

In (8), *PIEC*(*i*, *j*) = 1 means, e(j) is incident at v(i) and also participating in the MSP. Otherwise e(j) is ignored. Since, $ve(i,j) * x(j)$ is a binary variable, Equation (8) can be rewritten as,

$$PIEC(i,j) = ve(i,j) * x(j) \quad (9)$$

Since, the participating degree ***PD***(*i*) of v(*i*) is the sum of *PIEC*(*i*, *j*)'s from all *j*'s, ***PD***(*i*) can be expressed as,

$$PD(i) = \sum_{j=1}^{M} PIEC(i,j) \quad (10)$$

From (10) and (9),

$$PD(i) = \sum_{j=1}^{M} ve(i,j) * x(j) \quad (11)$$

The *PD*(*i*) given by (11) is used to specify the constraints on the participating degree of *v*(*i*).

IV. MOVING SINK PROBLEM

The purpose of the MS is to exchange data with sensors by physically visiting the location of the nodes. The following are the constraints imposed on the MS.

A. *constraints*

1) The MS should start the present tour at the starting node which may be the access point or the BS of the WSN.
2) It should visit every sensor node vertex exactly once in the present tour and should return to the starting node at the end of the present tour.
3) The intermediate non-sensor vertices should not repeat along the path travelled by MS.
4) The total travel distance (length) should be minimum.

Constraints 1 to 4 above are the standard requirements similar to that of the Travelling Salesman Problem. A repeated node (except the starting node) along the path creates sub-cycles and should be avoided. Additional constraints are imposed in our scheme to provide efficient movement of the MS as follows.

5) It should travel only along the grid lines of the graph. This constraint forces the MS tour path to be a rectangular polygon. (Here, the assumption is that proper physical paths are available for the MS to move along the grid lines).

   The MS Closed Path along the grid lines is abbreviated as MSCP.
6) The number of bends along the path should be minimum. In the proposed grid graph, the bends are at 90^0 and the MS has to spend additional energy and time to negotiate the bends. Travel with Minimum bends saves the energy and time for the MS.

The closed path travelled which satisfies the above constraints is referred as the *Minimum Bend Shortest Closed Path* (MBSCP). Determination of MBSCP satisfying these constraints is termed as the *Moving Sink Problem* (MSP).

B. *Objective*

The objective is to solve the MSP by determining MBSCP for the given graph. That is to find out the set of edges to be traversed by the MS to generate MBSCP.

C. *Approach towards the Solution*

The method adopted by us to solve the MSP is similar to solving TSP [12] in the sense that the proposed method also uses the Binary Integer Programming [13] as adopted in solving the standard TSP. But our main contribution is to convert the *minimum bend constraint* into a *linear one* and then adopt the binary integer programming technique.

V. FORMULATION OF CONSTRAINTS

We express the constraints of section IV in a proper algebraic form suitable for the Binary Integer Programming solver. To achieve this, we consider the various attributes of the MBSCP.

In the grid graph, MSCP is represented as a set of connected vertices forming the closed path It can also be represented by a set of participating edges that forms the closed path connecting all the sensor nodes. A typical MSCP on a grid graph would look as shown in Fig. 3, where MSCP is shown in red and the sensor nodes in blue.

The set of vertices that belong to MSCP is represented by the set ***MSCP_V*** which is a subset of ***V***. Similarly the set of edges which make up the ***MSCP*** is represented by the set ***MSCP_E*** which is a subset of ***E***. Thus, ***MSCP_V*** is the vertex set and ***MSCP_E*** is the edge set of MSCP.

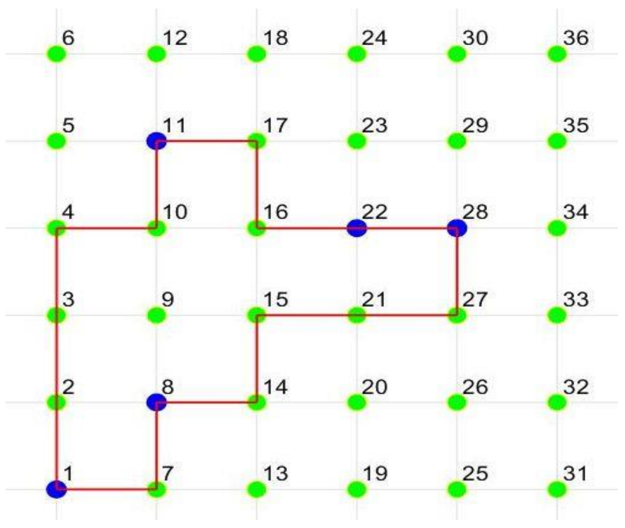

Fig. 3. A typical closed path with 16 participating nodes

A. Sensor Node Vertex Set

Let K be the number of *sensor nodes* deployed in the given grid based WSN. These sensor nodes are located at K grid points (vertices) of the graph. Let the vertex set occupied by the sensor nodes be represented as,

$$S = [s_{1,} s_2, \dots, s_u, \dots s_K] \tag{12}$$

Thus $s_u \in \boldsymbol{S}$, where $\boldsymbol{S}$ is a subset of $\boldsymbol{V}$. In the example of Fig. 3, the sensor nodes are marked in blue and $\boldsymbol{S}$ = [1, 8, 11, 22, 28] and K = 5. These vertices are the locations of the sensor nodes. Remaining vertices are non-sensor vertices. Since $\boldsymbol{V}$ is the entire set of vertices of the graph, the non-sensor vertices form the subset $\{\boldsymbol{V}-\boldsymbol{S}\}$. Thus, we have,

$\boldsymbol{S}$ = Sensor node vertex set
$\{\boldsymbol{V}-\boldsymbol{S}\}$ = Non-sensor node vertex set

B. Participating Degree of vertices belonging to the vertex set $\boldsymbol{S}$

The MSCP has to compulsorily pass through all the vertices in $\boldsymbol{S}$. Consider a segment of MSCP passing through s_u which belongs to $\boldsymbol{S}$. For example take the segment [17→11→10] from Fig. 3, where $v(11)$ belongs to $\boldsymbol{S}$. The path MSCP has to visit s_u exactly once. That is, the path should enter the vertex s_u only once and leave also only once. Thus the Participating Degree $\boldsymbol{PD}(s_u)$ of s_u (in the example case, $v(11)$) has to be 2. This rule holds true for all the sensor node vertices of $\boldsymbol{S}$. This requirement can be expressed as,

$$PD(s_u) = 2 \quad \forall\, s_u \in \boldsymbol{S} \tag{13}$$

This means,

$$PD(i) = 2 \quad \forall\, \mathrm{i} \in \boldsymbol{S} \tag{14}$$

From (14) and (11)

$$\sum_{j=1}^{M} ve(i,j) * x(j) = 2 \quad \text{for } i \in \boldsymbol{S} \tag{15}$$

Subset $\boldsymbol{S}$ is given and fixed. The basic constraint is that the MSCP to be determined must pass through all the vertices of $\boldsymbol{S}$ exactly once. This constraint, represented by (15), is basically a constraint on $x(j)$*'s* which are the decision variables.

C. Participating Degrees of vertices of subset $\{\boldsymbol{V}-\boldsymbol{S}\}$)

Consider a vertex $v(i) \in \{\boldsymbol{V}-\boldsymbol{S}\}$ which is same as $i \notin \boldsymbol{S}$. (In the example of Fig. 3, vertices belonging to $\{\boldsymbol{V}-\boldsymbol{S}\}$ are marked in green). Vertex $v(i)$ may or may not lie on the MSCP. (For example, in Fig. 3, $v(2)$ lies on the MSCP whereas $v(13)$ does not, even though both the vertices belong to $\{\boldsymbol{V}-\boldsymbol{S}\}$). Let us consider both the cases.

Case 1) Vertex $v(i)$ lies on the MSCP, that is $i \in$ ***MSCP_V.***

In this case, MSCP passes through $v(i)$ exactly once. Then, as explained earlier in the case of subset $\boldsymbol{S}$, the $\boldsymbol{PD}(i)$ of $v(i)$ is 2. That is,

$$\sum_{j=1}^{M} ve(i,j) * x(j) = 2 \qquad \forall\, i \in \boldsymbol{MSCP\_V} \tag{16}$$

Case 2) Vertex $v(i)$ does not lie on MSCP.

In this case, MSCP does not pass through the $v(i)$. Since the path neither enters nor leaves $v(i)$, the Participating Degree $\boldsymbol{PD}$(i) is 0. This constraint is expressed as,

$$\sum_{j=1}^{m} ve(i,j) * x(j) = 0 \qquad \forall\, i \notin \boldsymbol{MSCP\_V} \tag{17}$$

D. Admission Control Variable for vertices.

For each vertex $v(i)$, let us introduce the binary decision variable $y(i)$ such that,

$$y(i) = \begin{cases} 1 & if\ vertx\ i \in \boldsymbol{MSCP\_V} \\ 0 & if vertex\ i \notin \boldsymbol{MSCP\_V} \end{cases} \quad \forall i \in \boldsymbol{V} \tag{18}$$

From (16), (17) and (18), we see that Equation (16) holds true when $y(i) = 1$ and Equation (17) holds when $y(i) = 0$. Therefore, in the light of (18), Equations (16) and (17) can be represented by a single Equation as,

$$\sum_{j=1}^{m} ve(i,j) * x(j) = 2 * y(i) \qquad \forall i \in \boldsymbol{V} \tag{19}$$

From (18), we see that the value of $y(i)$ decides whether $v(i)$ is to be included in ***MSCP_V*** or not. Therefore $y(i)$'s form another set of decision variables to be determined by the optimization solver. The $y(i)$'s form the second decision vector $\boldsymbol{Y}$, {the first decision vector being $\boldsymbol{X}$ given by (7)}, as,

$$\boldsymbol{Y} = [y(1), y(2), \dots, y(i), \dots, y(N)] \tag{20}$$

From (18), we see that the ones of $\boldsymbol{Y}$ represent ***MSCP_V***. The optimal solution should satisfy the constraint (15) and (19).

E. Edge subset formats

A subset of edges in a graph can be represented in two formats. The conventional format and index format (bit mask format). In conventional format, the edge id's (edge numbers when edges are numbered) are used as the members of the subset. For example, consider the MSCP shown in Fig. 2. The MSCP is the path represented by ***MSCP_V***=[1, 2, 3, 4, 5, 6, 7, 8] and is shown in red.

The edge subset of MSCP is represented by ***MSCP_E***. The conventional format is represented as ***MSCP_EC***. In Fig. 2, ***MSCP_EC*** is,

MSCP_EC = [1, 2, 3, 6, 7, 8, 9, 10]

Here, the edges are arranged in the ascending order. The members of ***MSCP_EC*** are the corresponding edge numbers.

1) Index format of edge subset representation

The index format representation of ***MSCP_E*** is designated as ***MSCP_EI*** (symbol ***I*** is appended to indicate the index format). ***MSCP_EI*** is a binary vector of length ***M.*** The j^{th} element of ***MSCP_EI*** is set to 1 if edge *j* is a member of ***MSCP_EI***, else it is set to 0. That is the j^{th} element of ***MSCP_EI***, represented by *mscp_ei*(*j*) is set as,

$$mscp\_ei(j) = \begin{cases} 1 & if\ edge\ j\ \in\ \mathbf{MSCP\_EC} \\ 0 & if\ edge\ j\ \notin\ \mathbf{MSCP\_EC} \end{cases} \quad \forall j \in \boldsymbol{E} \qquad (21)$$

For example, in Fig. 2,
MSCP_EI = [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0]
The number of 1's in ***MSCP_EI*** is same as the number of edges in ***MSCP_EC***. Therefore, the number of edges in ***MSCP_EC*** represented by *L*, is given by,

$$L = \text{sum}(\boldsymbol{MSCP\_EI}) \qquad (22)$$

For example, In Fig.2,

L = sum(***MSCP_EI***) = sum([1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0]) = 8

In Matlab notation, ***MSCP_EC*** can be obtained from ***MSCP_EI*** as,

$$\boldsymbol{MSCP\_EC} = \boldsymbol{find}(\boldsymbol{MSCP\_EI}) \qquad (23)$$

In the index format representation, the main (full) set is the binary vector of all 1's.

F. Length of MSCP

The edge set of ***MSCP***, represented by ***MSCP_EC***, is a subset of ***E***. In our optimization method, the edges of ***MSCP_EC*** are selected based on the calculated value of the admission control variable *x*(*j*)'s as follows.

$$\left.\begin{array}{l} j \in \boldsymbol{MSCP\_EC} \quad if\ x(j) = 1 \\ j \notin \boldsymbol{MSCP\_EC} \quad if\ x(j) = 0 \end{array}\right\} \qquad (24)$$

Thus *x*(*j*) acts as the admission criterion to include edge *j* as a member of ***MSCP_EC***. From (21) and (24) we see that *mscp_ei*(*j*) and *x*(*j*) are same for ∀*j* ∈ ***E***. Therefore

$$\boldsymbol{MSCP\_EI} = \boldsymbol{X} \qquad (25)$$

Therefore, from (25) and (22), the length of MSCP in terms of the number of edges is,

$$L = sum(X) = \sum_{j=1}^{M} x(j)$$

Since *L* is a function of the decision variable *X*, we use *L*(***X***) instead of just *L*. Therefore the above equation is rewritten as,

$$L(X) = \text{sum}(\mathbf{X}) = \sum_{j=1}^{M} x(j) \qquad (26)$$

In solving MSP, the first objective function is L(X).

G. Minimum Bend Paths

Consider two different paths starting from vertex 1 and ending with vertex 16 as shown in Fig. 4. Path ***P***1 and ***P***2 are made up of vertices as,

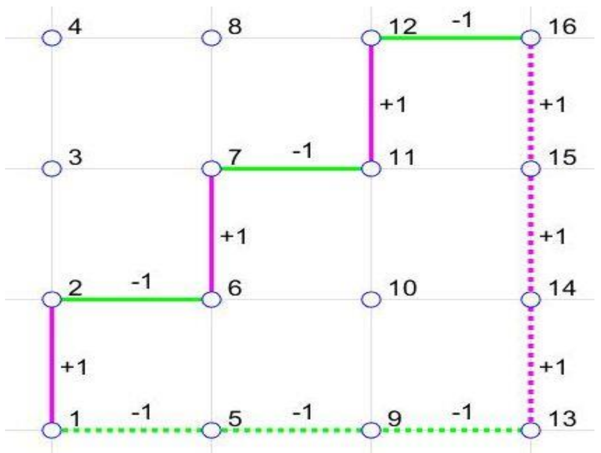

Fig. 4. Two different paths having the same length, but different number of bends

P1 = [1, 2, 6, 7, 11, 12, 16]

P2 = [1, 5, 9, 13, 14, 15, 16]

Both ***P***1 and ***P***2 have the same length 6. But the number of bends in *P*1 and *P*2 are different. ***P***1 has 5 bends while ***P***2 has only one bend. Now, consider the positions of vertical and horizontal edges in ***P***1 and ***P***2. Horizontal edges in green are represented by 'g' and vertical edges in magenta are represented by 'm'. Then, in terms of horizontal and vertical edges,

P1 = [m, g, m, g, m, g]

P2 = [g, g, g, m, m, m]

Now, consider any two adjacent edges along the path. A bend occurs if the two adjacent edges have different orientations (one horizontal and the other vertical) and there is no bend if the two adjacent edges have the same orientation (both horizontal or both vertical). To distinguish the horizontal and vertical edges we introduce the *Edge Orientation Index* which is a numerical representation of the edge orientations. The numerical representation provides an easy way to calculate the number of bends along the MSCP.

H. Edge Orientation Index

Edge Orientation Index (*eoi*) of edge *j* is defined as

$$eoi(j) = \begin{cases} +1 & \text{if edge } j \text{ is vertical} \\ -1 & \text{if edge } j \text{ is horizontal} \end{cases} \quad \forall\, j \in \boldsymbol{E} \qquad (27)$$

That is, the orientation of vertical edges is represented by +1 whereas that of the horizontal edges by −1. For example, the *eoi*'s are marked along the edges of Fig. 4.

I. Edge Orientation Index Vector for the entire grid graph

The orientation of edge e(*j*) is given by (3) and is known for the edges in the given grid graph. Then from (27) we can calculate the eoi's for all the edges of the graph. The collection of eoi's form the *Edge Orientation Index Vector* (***EOIV***) as,

$$\boldsymbol{EOIV} = [eoi(1), eoi(2),\ldots,eoi(j), \ldots, eoi(M)] \tag{28}$$

The size of ***EOIV*** is 1x*M* and it is a vector of +1's and −1's. For example, the ***EOIV*** vector for the graph of Fig. 2, is given in Table II.The first row gives the edge number

TABLE II
EOIV FOR THE GRID GRAPH OF FIG. 2.

E	1	2	3	4	5	6	7	8	9	10	11	12
EOIV	+1	+1	−1	−1	+1	−1	+1	−1	−1	−1	+1	+1

J. *Formation of bends along a closed path*

Consider a vertex *v*(*i*) lying along MSCP. Now the participating edges are those edges of *v*(*i*) which lie on MSCP. Here the participating degree ***PD***(*i*) = 2. The two participating edges incident on the vertex can form four possible combinations of bends and two bend-free combinations as shown in Fig. 5.

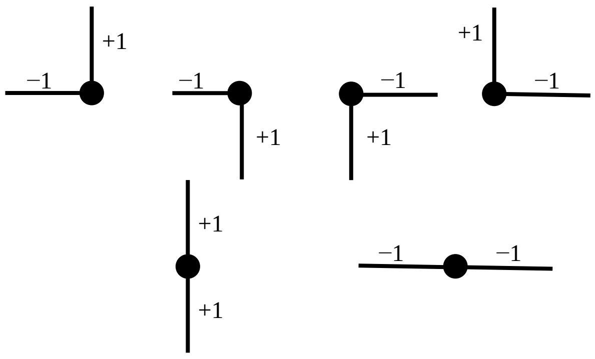

Fig. 5. Formation of bends at a vertex

From Fig. 5, we see that for the two participating incident edges forming a bend (corner), the *eoi*'s are [−1, +1] or [+1, −1]. But, for a bend-free vertex (junction), the *eoi*'s are [−1, −1] or [+1,+1]. An important conclusion from these observations is, a vertex with two incident edges, the sum of eoi's is 0 if the vertex hosts a bend. Else the sum of eoi's is ±2. This fact is used to detect the presence of a bend at a vertex. The absolute value of the sum of the eoi's of the two incident edges decides the existence of a bend at that vertex.

Consider vertex *v*(*i*) which is a member of ***MSCP_V***. Let the two relevant participating incident edges forming the set be represented as ***PE***(i) = [*e*(*i*,1) and *e*(*i*,2)]. Edge Orientation Indices of these two edges are represented as $eoi\{e(i,1)\}$and $eoi\{e(i,2)\}$. Then, the absolute value, abs(*eoi*{(*e*(*i*, 1)} + *eoi*{*e*(*i*, 2)}) decides the presence of a bend at that vertex. This deciding value is called the *Vertex Bend Index* (*vbi*) of that vertex.

Then the *vbi*(*i*) of *v*(*i*) is defined as,

$$vbi(i) = abs(eoi\{e(i,1)\} + eoi\{e(i,2)\}) \tag{29}$$

Then, a bend at *v*(*i*) depends on *vbi*(*i*) as,

$$\text{Bend at } v(i) \text{ does } \begin{cases} \text{exist} & \text{if } vbi(k) = 0 \\ \text{not exist} & \text{if } vbi(k) = 2 \end{cases} \tag{30}$$

Equation (30) means,

$$\text{No. of bends at } v(i) = \begin{cases} 1 & \text{if } vbi(k) = 0 \\ 0 & \text{if } vbi(k) = 2 \end{cases} \tag{31}$$

From (31), we can express the number of bends at *v*(*i*), represented by *nbv*(i) as,

$$nbv(i) = \frac{2 - vbi(i)}{2} = 0.5 * (2 - vbi(i)) \tag{32}$$

K. *Total number of bends in* MSCP

The MSCP passes through the vertex set ***MSCP_V***. The corresponding edge set ***MSCP_EC*** is determined according (24) and ***MSCP_ EI*** is same as the admission control vector ***X*** as given in (25). Number of bends, *nbv*(i), at an individual vertex is given by (32). Therefore the total number of bends in MSCP is given by the summation of *nbv*(i)'s for those *i*'s belonging to ***MSCP_V***. That is, the *Total Number of Bends* (*TNB*) can be expressed as,

$$TNB = \sum_{i \in \boldsymbol{MSCP\_V}} nbv(i) \tag{33}$$

Here, *nbv*(i) is the number of bends at vertex *v*(*i*) as given by (32). Substituting for *nbv*(i) in (33) from (32), we get,

$$TNB = \sum_{i \in MSCP\_V} 0.5 * (2 - vbi(i))$$

The RHS is simplified to get,

$$TNB = \sum_{i \in \boldsymbol{MSCP\_V}} 1 - 0.5 * \sum_{i \in \boldsymbol{MSCP\_V}} vbi(i)$$

This can be rewritten as,

$$TNB = |\boldsymbol{MSCP\_V}| - 0.5 * \sum_{k \in MSCP\_V} vbi(i) \tag{34}$$

From (18) and (20),

$$|\boldsymbol{MSCP\_V}| = \text{sum}(\boldsymbol{Y}) \tag{35}$$

From (34) and (35),

$$TNB = \text{sum}(\boldsymbol{Y}) - 0.5 * \sum_{k \in MSCP\_V} vbi(i) \tag{36}$$

Since vertices outside ***MSCP_V*** have no incident edges, they do not contribute to *vbi*(i). Hence, when *i* in (36) is extended to all edges of the grid, $\sum_{i \in MSCP_V} vbi(i)$ remains same. Therefore when the range of i is extended to I ∈ ***V*** the RHS of (36) remains same. Therefore, (36) can be rewritten as,

$$TNB = \text{sum}(\boldsymbol{Y}) - 0.5 * \sum_{i=1}^{N} vbi(i) \tag{37}$$

L. *Determination of Vertex Bend Index, vbi*(*i*)

From (29), $vbi(i) = abs(eoi\{e(i,1)\} + eoi\{e(i,2)\})$. To determine *vbi*(i), we should find e(*i*, 1) and e(*i*, 2) which are the edges incident on *v*(*i*) and also lie on the MSCP. The resulting participating edge set ***PE***(*i*) is,

$$\boldsymbol{PE}(i) = [e(i, 1), e(i, 2)] \tag{38}$$

Here [e(i, 1), e(i, 2)] belong to the edge set ***MSCP_EC*** and also incident on $v(i)$. This condition is represented as,

$$\boldsymbol{PE}(i) \subseteq \boldsymbol{MSCP\_EC} \quad (39)$$

Now, represent the set relation (39) in the index format as,

$$\boldsymbol{PEI}(i) \subseteq \boldsymbol{MSCP\_EI} \quad (40)$$

Here, $\boldsymbol{PEI}(i)$ is the index format representation of $\boldsymbol{PE}(i)$.

For example in Fig. 2, for vertex v(5), the value of $\boldsymbol{PE}(5) = [7, 9]$ and $\boldsymbol{PEI}(5) = [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0]$.

From (40) and (25),

$$\boldsymbol{PEI}(\mathrm{i}) \subseteq \boldsymbol{X} \quad (41)$$

Full set of the edges of $v(i)$ are given by the 1's of row i of edge-vertex matrix ***EV***. (See property 3 of ***EV*** matrix, Section III. D). Thus row i represented by $\boldsymbol{EV}(i, :)$ gives the set of all edges of vertex $v(i)$. Therefore, $\boldsymbol{PEI}(i)$ is a sub set of $EV(i, :)$. This constraint is expressed as,

$$\boldsymbol{PEI}(\mathrm{i}) \subseteq \boldsymbol{EV}(i, :) \quad (42)$$

Both (41) and (42) are to be satisfied by $\boldsymbol{PEI}(i)$. Therefore, it is given by the intersection of the two sets X and EV(i, :) as,

$$\boldsymbol{PEI}(i) = \boldsymbol{X} \cap \boldsymbol{EV}(i, :) \quad (43)$$

In (43), all the terms are the subsets of ***E*** and are in the index form. Since X and $\boldsymbol{EV}(i\,:)$ are in the binary vector format with 1's representing the set elements, the intersection X∩ EV(i, :) can be represented by the logical AND of them as,

$$\boldsymbol{X} \cap \boldsymbol{EV}(i, :) = \boldsymbol{X} \wedge \boldsymbol{EV}(i, :) \quad (44)$$

The logical operation can be converted to arithmetic operation as,

$$\boldsymbol{X} \wedge \boldsymbol{EV}(i, :) = \boldsymbol{X}.*\ \boldsymbol{EV}(i, :) \quad (45)$$

The RHS of (44) is the Matlab notation for the element wise product of two vectors.
From (43), (44) and (45),

$$\boldsymbol{PEI}(\mathrm{i}) = \boldsymbol{X}\ .*\ \boldsymbol{EV}(i, :) \quad (46)$$

For the example of Fig. 2, the values X, EV(5, :) and ***PEI***(5) are shown in Table III.

TABLE III.
VALUES OF ***X***, ***EV***(5,:), ***PEI***(5) AND ***EOI***(5)

	1	2	3	4	5	6	7	8	9	10	11	12
X	1	1	1	0	0	1	1	1	1	1	0	0
EV(5, :)	0	0	0	1	1	0	1	0	1	0	0	0
PEI(5)	0	0	0	0	0	0	1	0	1	0	0	0

2) Determination of eoi's of e(i, 1) *and e(i,2)*

The Edge Orientation Index Vector, ***EOIV*** for the given graph is known and is given by (28). ***EOIV*** gives the orientation of all the edges of the full edge set ***E***. That is, $eoi(j)$ gives the orientation of edge e(j) whose index location is j in ***E*** for all j's from j = 1 to M. But we have to select only those two eoi's of edges which are specified by $\boldsymbol{PEI}(i)$. The position of these two edges e(i, 1), e(i, 2) are given by $\boldsymbol{PEI}(i)$ in the index form. To select the corresponding eoi's, we use the bit-mask technique as,

$$[\ eoi\{e(i,1)\}.\ \ eoi\{e(i,2)\}\ ] = \boldsymbol{PEI}\,(i).*\boldsymbol{EOIV} \quad (47)$$

The element wise multiplication selects those eoi's of ***EOIV*** for which the index locations e(i, 1) and e(i, 2) are ones in $\boldsymbol{PEI}(i)$,. Thus the respective eoi's are stored in ***PE***I(i).****EOIV***. Let us designate the LHS of (47) by $\boldsymbol{EOI}(i)$ as,

$$\boldsymbol{EOI}(i) = [\ eoi\{e(i,1)\}.\ \ eoi\{e(i,2)\}\ ] \quad (48)$$

Then, from (47) and (48),

$$\boldsymbol{EOI}(i) = \boldsymbol{PEI}(i).*\boldsymbol{EOIV} \quad (49)$$

From (46) and (49),

$$\boldsymbol{EOI}(i) = \boldsymbol{PEI}(i).*\boldsymbol{EOIV} = \boldsymbol{X}.*\boldsymbol{EV}(i, ;).*\boldsymbol{EOIV} \quad (50)$$

Here, $\boldsymbol{EOI}(i)$ is a binary vector of size 1xM.

In (50), ***EOIV*** and ***EV***(i, :) are known and constants for the given grid graph. Vector ***X*** is the decision variable to be determined by the optimization solver. Equation (50) expresses the Edge Orientation information in terms of ***X***. For the example of Fig. 2, for k = 5,

$\boldsymbol{X}$ = [1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0],

From Table I,

$\boldsymbol{EV}(5, :)$ = [0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0]

From Table II,

$\boldsymbol{EOIV}$ = [+1, +1, −1, −1, +1, −1, +1, −1, −1, +1, −1, +1]

From (50)

$$\boldsymbol{EOI}(5) = [1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0,\ 0].*$$
$$[0,\ 0,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0].*$$
$$[+1,\ +1,\ -1,\ -1,\ +1,\ -1,\ +1,\ -1,\ -1,\ +1,\ -1,\ +1]$$

EOI(5) = [0, 0, 0, 0, 0, 0, +1, 0, −1, 0, 0, 0] .

From (29), we know that,

$$vbi(i) = abs\big(sum([eoi\{e(i,1)\}, eoi\{e(i,2)\}])\big)$$

From (29) and (47),

$$vbi(i) = abs(\mathrm{sum}(\boldsymbol{PEI}(\mathrm{i}).*\ \boldsymbol{EOIV}\ )) \quad (51)$$

From (51) and (50)

$$vbi(i) = abs\big(sum(\mathbf{X}.*\ \mathbf{EV}(i,:).*\ \mathbf{EOIV})\big) \quad (52)$$

Substituting for $vbi(i)$ in (37) from (52), we get,

$$TNB =$$

$$\mathrm{sum}(\boldsymbol{Y}) - 0.5 * \sum_{i=1}^{N} abs\big(sum(\boldsymbol{X}.*\ \boldsymbol{EV}(i,:).*\ \boldsymbol{EOIV})\big) \quad (53)$$

Since TNB is a function of ***X*** and ***Y***, it is represented as,

$$TNB(\boldsymbol{X},\boldsymbol{Y}) =$$

$$\mathrm{sum}(\boldsymbol{Y}) - 0.5 * \sum_{i=1}^{N} abs\big(sum(\boldsymbol{X}.*\ \boldsymbol{EV}(i,:).*\ \boldsymbol{EOIV})\big) \quad (54)$$

From (54), we see that $TNB(\boldsymbol{X},\boldsymbol{Y})$ is a linear function of decision variables ***X*** and ***Y.*** The derivation of formula (54) is the main contribution of this work.

In solving for ***MBSCP***, we have to minimize both the total length $L(\boldsymbol{X})$ of the M***BSCP*** given by (26) and also minimum bend term $TNB(\boldsymbol{X},\boldsymbol{Y})$ as given by (54). Thus MBSCP is a Bi-objective Minimization.

VI. Formulation of MBSCP.

The scalarized objective function for solving the MBSCP designated by $F(\boldsymbol{x})$ is taken as,

$$\text{Minimize} \qquad F(\boldsymbol{X}) = L(\boldsymbol{X}) + \lambda * \mathrm{TNB}(\boldsymbol{X},\boldsymbol{Y}) \qquad (55)$$

The scalarizing parameter λ is experimentally determined to give the best result. The optimization problem is, to minimize $F(\boldsymbol{X})$ given by (55) subjected to the constraints specified by (15) and (19) which are repeated here. The constraints are,

$$\sum_{j=1}^{M} ve(i,j) * x(j) = 2 \quad \text{for } i \in \boldsymbol{S}$$

$$\sum_{j=1}^{m} ve(i,j) * x(j) = 2 * y(i) \qquad \forall i \in \boldsymbol{V}$$

over $i = 1$ to N. Here x(j)'s and y(i)'s are the binary decision variables for edges and vertices respectively. These are determined by the Binary Integer Program solver. The process of finding the optimal ***MBSCP*** in this way is referred as the ***MBSCP*** method.

A. *Selection of the scalarizing parameter* λ

Scalariziation converts the bi-objective optimization into single weighted objective one. When $\lambda = 0$, only minimum length objective is satisfied. TNB will be relatively high. When λ increases, more weightage is given to the minimum bends criterion and *TNB* decreases. However, certain upper bound exists for λ and if it is increased beyond that threshold, the ***MBSCP*** does not converge. A judicious value for λ is chosen experimentally by solving ***MBSCP*** for different values of λ. In our examples, we found that $\lambda = 0.3$ minimizes *TNB* with fast convergence. Minimization of $F(\boldsymbol{X})$) is carried out using the binary integer programming.

VII. Simulation Results

Example1.

Here, W = 8 and H= 9. The number of vertices are N = 72 and the number of edges are M = 127. The number of sensor nodes is 17 and the sensor node vertex set ***S*** is,

S = [1, 8, 11, 22, 28, 32, 34, 38, 42, 47, 48, 52, 54, 57, 58, 60, 63];

Sensor node vertices are marked in blue.

The optimal ***MBSCP*** is determined for $\lambda = 0$, 0.15 and 0.3. The Length of the path and TNB are shown in Table IV.

The corresponding ***MBSCP*** paths are shown in red in Fig. 6, Fig. 7 and Fig. 8 respectively. From simulation result of Table IV, an important observation is that the length of the optimal ***MBSCP*** remains same as λ varies.

TABLE IV.
VALUES OF LENGTH AND ***TNB*** OF THE ***MSCP***

Scalarizer λ	Length of ***MBSCP***	***TNB*** of ***MBSCP***
0.00	38	22
0.15	38	20
0.30	38	16

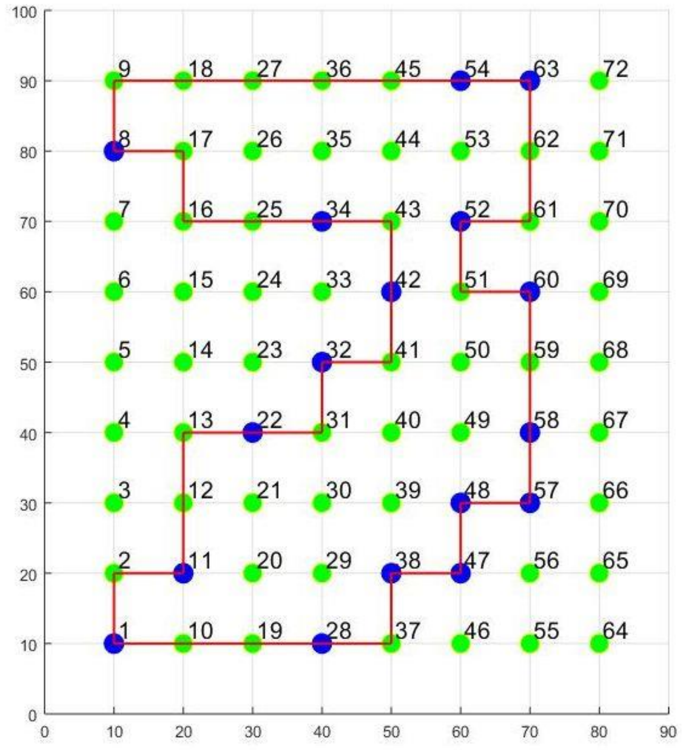

Fig. 6. Optimal MSP with λ = 0 and TNB = 22

VIII. Comparison with other methods

Clarkson's method [9] determines the minimum bend shortest rectilinear path. Therefore, initially, the ***MSCP*** is solved without considering the minimum bend criterion. Once the target points of the path are obtained, Clarkson's method is applied to determine the minimum bend shortest paths between successive points on the path. Another method by Basagni [10] uses MILP to get the optimal ***MSCP***. But in [10], minimum bend criterion is not discussed. Therefore an additional method is needed to minimize the number bends. Compared to Clarkson's and Bagsani's two stage processes, ***MBSCP*** is a single stage integrated process and therefore takes less time.

The comparison of time consumed by Clarkson's and Bagsani's method with the proposed minimum bend shortest closed path, ***MBSCP*** is shown in Fig. 9. Here, the number of grid points *N*, is incremented in multiples of 50. In each case,

the number of sensor nodes K, present is set to 10 % of N. The location of the sensor node grid points are selected randomly over the grid graph.

From Fig. 9, it can be seen that for smaller number of grid points, ***MBSCP*** is better compared to the other two methods. At higher number of nodes, all of them have almost the same execution time.

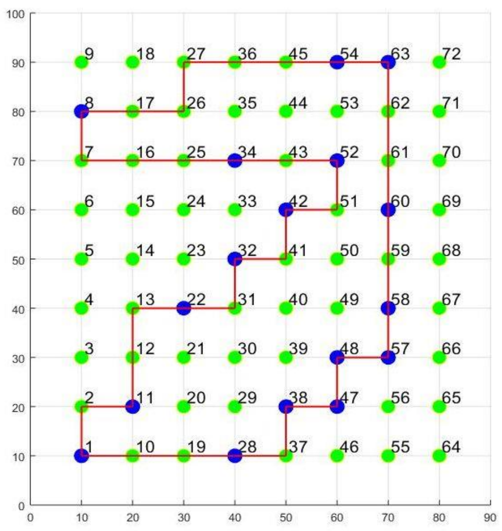

Fig. 7. Optimal MSP with λ = 0.15 and TNB = 20

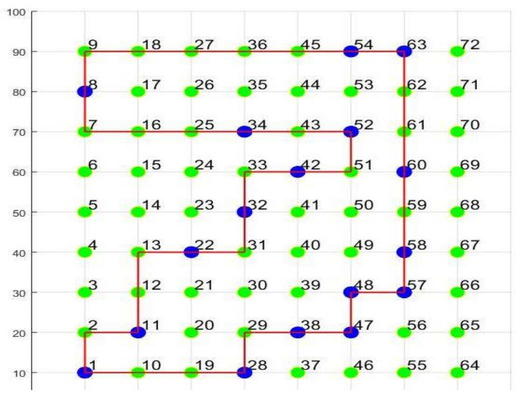

Fig. 8. Optimal MSP with λ = 0.3 and TNB = 16

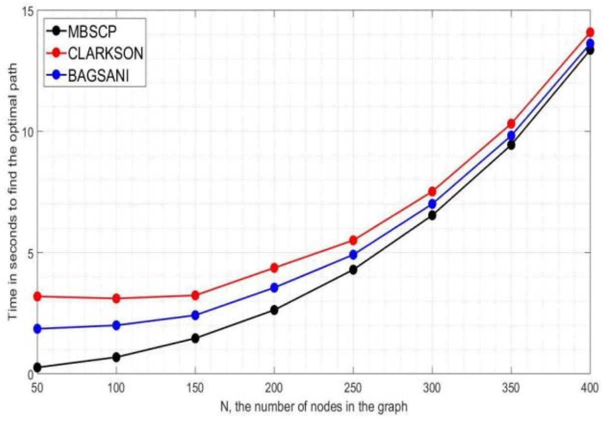

Fig. 9. Execution time versus N, the number of nodes

IX. Conclusion

A new technique is presented to determine the minimum bend shortest rectilinear path for a moving sink in a Wireless Sensor Network. The total number of bends along the path is expressed as a linear function of the decision variables. Then the linear integer program is used to solve the optimization problem. In this method, both the shortest path and the minimum bend criterion are met simultaneously. This integrated approach is a novel and unique solution to solve the moving sink path problem in a wireless sensor network.

References

[1] C. Tunca, S. Isik, M. Y. Donmez and C. Ersoy, "Distributed Mobile Sink Routing for Wireless Sensor Networks: A Survey," in IEEE Communications Surveys & Tutorials, vol. 16, no. 2, pp. 877-897, Second Quarter 2014. DOI: 10.1109/SURV.2013.100113.00293.

[2] M. Di Francesco, S. K. Das, and G. Anastasi, "Data collection in wireless sensor networks with mobile elements: A survey," ACM Trans. Sensor Networks, vol. 8, no. 1, pp. 1–31, 2011. DOI: 10.1145/1993042.1993049.

[3] W. Liang, J. Luo, and X. Xu, "Prolonging network lifetime via a controlled mobile sink in wireless sensor networks," in Global Commu -nications Conf. (GLOBECOM 2010), IEEE, 2010, pp. 1 –6. DOI: 10.1109/GLOCOM.2010.5683095.

[4] Z. Wang, S. Basagni, E. Melachrinoudis, and C. Petrioli, "Exploiting sink mobility for maximizing sensor networks lifetime," in Proc. of the 38th Annual Hawaii Int. Conf. on System Sciences (HICSS '05), 2005, pp.03-06. DOI: 10.1109/HICSS.2005.259.

[5] Huang, Hailong. (2017). Performance Improvement by Introducing Mobility in Wireless Communication Networks. eprint arXiv:1712.02436, 12/2017, 2017.

[6] V. G. Deineko , B. Klinz , A. Tiskin , G. J. Woeginger "Four-point Conditions for the TSP," Journal Discrete Optimization, Vol. 14 Issue C, November 2014 pp. 147-159. DOI: 10.1016/j.disopt.2014.09.003.

[7] A. Maheshwari, J.R. Sack, and H.N. Djidjev. Link distance problems, in: J.R. Sack and J. Urrutia, editors, Handbook of Computational Geometry, pp. 519-558. El- sevier Science Publishers B.V. North Holland, Amsterdam, 2000.

[8] 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., 2 (1992), pp. 61−74. DOI: 10.1142/S0218195992000056.

[9] 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., pp 251−257, 1987. DOI: 10.1145/41958.41985.

[10] S. Basagni, A. Carosi, C. Petrioli, Mobility in wireless sensor networks, in: Algorithms and Protocols for Ad Hoc and Sensor Networks, John Wiley & Sons, Inc., 2007.

[11] D. P. Wagner, R. S. Drysdale, C. Stein An O(n 5/2 log n) algorithm for the rectilinear minimum link-distance problem in three dimensions. Comput. Geom. Theory Appl. 42(5) (2009) 376–387

[12] Diaby, M., "The Travelling Salesman Problem: A Linear Programming Formulation", WSEAS Transactions on Mathematics, issue 6, vol. 6, pp. 745-754, 2007.

[13] Pataki, G., 2003. Teaching Integer Programming formulations using the traveling salesman problem. SIAM Review 45 (1), 116–123.