Step 1: Graph with Nodes and Edges

(1) ---- (2)

| |

(3) ---- (4) ---- (5)

Step 2: Determine Grid Boundaries

Coordinates:

(1, 0, 0), (2, 2, 2), (3, 3, 1), (4, 5, 4), (5, 7, 3)

minX = 0, minY = 0, maxX = 7, maxY = 4

Step 3: Create Grid

(0,0) (1,0) (2,0) (3,0) (4,0) (5,0) (6,0) (7,0)

(0,1) (1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,1)

(0,2) (1,2) (2,2) (3,2) (4,2) (5,2) (6,2) (7,2)

(0,3) (1,3) (2,3) (3,3) (4,3) (5,3) (6,3) (7,3)

(0,4) (1,4) (2,4) (3,4) (4,4) (5,4) (6,4) (7,4)

Step 4: Calculate Coverage

View Range = 3.0

For each grid point (x,y):

Count nodes within distance <= view range

Example for (3,2):

Covered Nodes: (2), (3), (4)

Total Covered: 3 nodes

Step 5: Find Optimal Position

Compare coverage for all grid points

Optimal Position: (4.5 , 2.0) for this point nodes coverd are (2),(3),(4),(5)

Explanation:

1. Grid Boundaries:

◦ Determine the minimum and maximum x and y coordinates among the nodes to set the grid boundaries.

2. Grid Traversal:

◦ Use nested loops to traverse each point on the grid within the determined boundaries, with a specified step size (gridStep).

3. Distance Calculation:

◦ Use the Euclidean distance to determine the proximity of nodes to each grid point.

4. Find Optimal Position:

◦ Iterate through all grid points.

◦ Track the grid point that covers the maximum number of nodes.

◦ If multiple points have the same coverage, choose the first one found during the traversal.

2. Secondary Criterion:

◦ Alternatively, you could add a secondary criterion like proximity to the center.

Nodes and Their Cartesian Coordinates

Assuming we already have the converted Cartesian coordinates:

Node 1: (6371, 0)

Node 2: (6370.15, 222.39)

Node 3: (6368.79, 111.19)

Node 4: (6367.12, 444.78)

Node 5: (6364.24, 333.59)

Edges and Their Weights

Edges (with their Euclidean distances calculated as weights):

Edge (1, 2): Distance = 222.4

Edge (2, 3): Distance = 111.2

Edge (3, 4): Distance = 334.0

Edge (4, 5): Distance = 334.1

Edge (1, 3): Distance = 111.2

Edge (2, 4): Distance = 223.2

Edge (3, 5): Distance = 223.2

Example: Shortest Path from Node 1 to Node 5

Initialization

Node Distance from Node 1 Previous Node Priority Queue

1 0 - (1, 0)

2 ∞ -

3 ∞ -

4 ∞ -

5 ∞ -

Iteration 1 (Node 1)

Current Node: 1

Edges from Node 1: (1, 2), (1, 3)

Update distances for Nodes 2 and 3

Node Distance from Node 1 Previous Node Priority Queue

1 0 - (2, 222.4), (3, 111.2)

2 222.4 1

3 111.2 1

4 ∞ -

5 ∞ -

Iteration 2 (Node 3)

Current Node: 3

Edges from Node 3: (3, 2), (3, 4), (3, 5)

Update distances for Nodes 4 and 5

Node Distance from Node 1 Previous Node Priority Queue

1 0 - (2, 222.4), (4, 445.2), (5, 334.4)

2 222.4 1

3 111.2 1

4 445.2 3

5 334.4 3

Iteration 3 (Node 2)

Current Node: 2

Edges from Node 2: (2, 1), (2, 3), (2, 4)

Update distance for Node 4 (if shorter path found)

Node Distance from Node 1 Previous Node Priority Queue

1 0 - (4, 445.2), (5, 334.4)

2 222.4 1

3 111.2 1

4 445.2 3

5 334.4 3

Iteration 4 (Node 5)

Current Node: 5

Destination node reached, algorithm stops.

Shortest Path Reconstruction

From the previous table:

Start at Node 5: Previous Node is 3

Move to Node 3: Previous Node is 1

Move to Node 1: Start Node

Shortest path: 1 -> 3 -> 5

Consider a graph with nodes A, B, C, and D, and edges with the following weights:

A -> B with weight 1

A -> C with weight 4

B -> C with weight 2

B -> D with weight 5

C -> D with weight 1

If the start node is A and the destination node is D:

Initialization:

distances: {A: 0, B: ∞, C: ∞, D: ∞}

previous: {A: null, B: null, C: null, D: null}

pq: [(A, 0)]

Processing Nodes:

A is processed first. Adjacent nodes B and C are updated:

distances: {A: 0, B: 1, C: 4, D: ∞}

previous: {A: null, B: A, C: A, D: null}

pq: [(B, 1), (C, 4)]

B is processed next. Adjacent nodes C and D are updated:

distances: {A: 0, B: 1, C: 3, D: 6}

previous: {A: null, B: A, C: B, D: B}

pq: [(C, 3), (C, 4), (D, 6)]

C is processed next. Adjacent node D is updated:

distances: {A: 0, B: 1, C: 3, D: 4}

previous: {A: null, B: A, C: B, D: C}

pq: [(D, 4), (D, 6)]

Termination:

When D is processed with a distance of 4, the algorithm terminates as the shortest path to the destination has been found.