Hole Healing in Mobile Sensor Network

This is a joint work by:
Pritam Goswami, Sangita Patra, Buddhadeb Sau
Department of Mathematics
Jadavpur University, Kolkata, India

Outline

Introduction

Problem Introduction

Coverage Hole: Definition

An area or a point inside a sensing network which is failed to be sensed by all the sensors in the network is called a coverage hole.

Problem Introduction

Coverage Hole: Definition

An area or a point inside a sensing network which is failed to be sensed by all the sensors in the network is called a coverage hole.

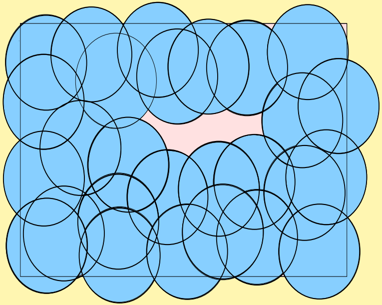


Figure: Hole

Problem Introduction

Why healing of hole is necessary

A hole in the sensing network means,

- Lack of monitoring.
- Disrupted functionality of the network.

Problem Introduction

Problem Definition

- $\mathcal{F}$ be a field.
- $\#$ active mobile sensor nodes $= k$.
- coverage hole exists.
- k nodes are sufficient to cover $\mathcal{F}$.

The problem is to design an algorithm such that the hole is covered by rearranging the nodes.

Earlier Works

- In [Mahboubi and Aghdam, 2017] an voronoi diagram and virtual force based algorithm for maximum coverage.
- in [Li and Hunter, 2008] an covering algorithm has been proposed but it fails for trivial holes.

Novelty

Our algorithm is:

1. effective with minimum number of active nodes to cover holes.
2. Robust against multiple coverage hole.
3. in best case heals hole within $O(n)$ time and minimal movements.

Model

Sensor nodes :

- connectivity radius r.
- Sensing radius s.
- are homogeneous.
- has limited memory.
- has agreement on global co ordinate system.
- can exchange information with other nodes within $r(\leq 2s)$ distance.

Preliminaries

Result 1: ([Sau and Mukhopadhyaya, 2013])

- $\mathcal{N}$: network deployed on a field of interest $\mathcal{F}$.
- $\phi(v)$: co ordinate of node v on $\mathcal{F}$.
- $\overline{sz}(\phi(v))$: Perimeter of the circle with center at $\phi(v)$ and radius s.
- $\mathcal{F}$ is sensing covered if and only if:
  - $\forall\ v$, each point on $\overline{sz}(\phi(v))$ is within the sensing zone of at least one other node.

Preliminaries

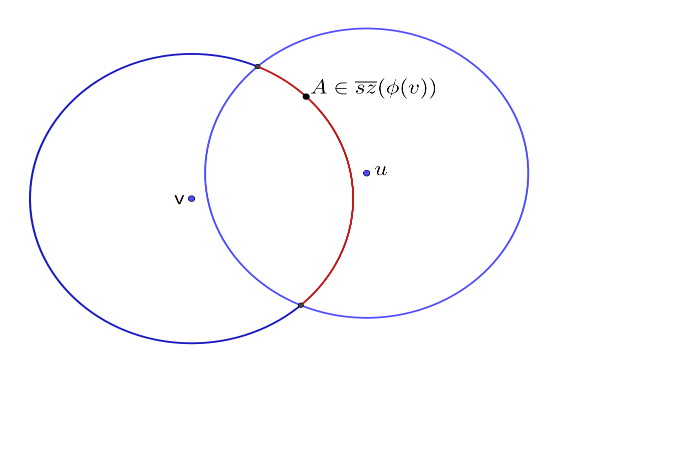


Figure: Coverage of boundary point

Motivation

Result 2: ([Zhang and Hou, 2005])

All sensor nodes:

1. completely cover a region $\mathcal{F}$.
2. homogeneous

Then, minimizing # working nodes ≡ minimizing overlap .

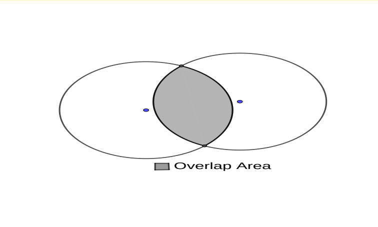


Figure: overlap

Motivation

Result 3: ([Zhang and Hou, 2005])

To cover one crossing point of two disks with minimum overlap:

- only one disk used.
- centers of the three disk should form a equilateral triangle.
- side length $\sqrt{3}s$. (radius of the disks : s)

Motivation

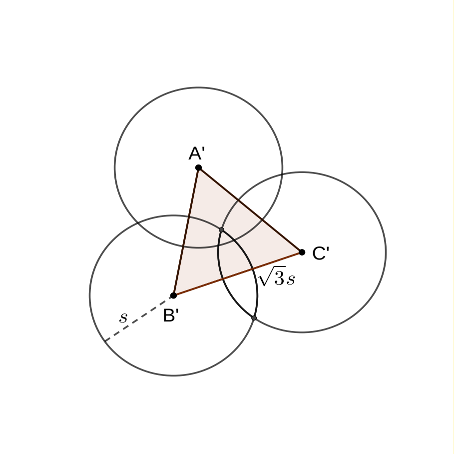


Figure: minimum overlap

Motivation

Intuition for the coverage

To minimize the number of nodes to heal the coverage we have rearranged the nodes such that they form a **hexagonal grid configuration** where each grid unit is $\sqrt{3}s$ (s is the sensing range of the nodes).

Motivation

Hexagonal Grid Configuration: Definition

Each node has exactly six neighbors on a regular hexagon with fixed length sides.

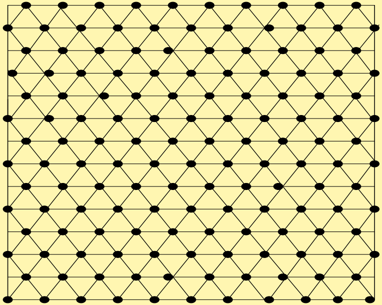


Figure: Hexagonal Configuration

Hole Healing Method

Data Structure

GridEntry

GridEntry {/*Data type to store the information of a grid point*/

pos: represents the position of the grid point.

assignedNode: id of the node assigned; initially 0.

}

Data Structure

NodeInfo

NodeInfo	{/*Data type for holding the information a node*/	
	id:	the unique identity of a node
	initLoc:	location of the node as per deployment.
	loc:	reconfigured location; initially 0 which is not a valid position of a node
}		

Algorithm

Algorithm 1: assignNodesToGrid ($\mathcal{G}(\mathcal{F})$, GF)

input: $\mathcal{G}(\mathcal{F})$, a 6-neighbor grid with grid unit $l = \sqrt{3}s$ of the monitoring region $\mathcal{F}$ and an array GF contains relevant information of the grid.

output: hexagonal configuration of the network nodes.

Global variables: array $GF[N]$ of type $GridEntry$, N = number of grid points, array $nodes[n]$ of type $NodeInfo$, n = number of deployed nodes.;

for $i \leftarrow 0$ **to** n **do**

  Call $assignNode(id)$; // id assigned to the exact grid location

Algorithm

Algorithm 2: assignNode(id)

output: assigns the node $nodes[id]$ **to a grid point**;
while *(there is any unvisited grid point within distance 2s from* $nodes[id].initLoc$*)* **do**
    Let, x be a closest unvisited grid point within distance $2s$ from $nodes[id].initLoc$;
    if $GF[x].assignedNode = 0$ **then**
        Set $nodes[id].loc \leftarrow x$ and $GF[x].assignedNode \leftarrow id$;
        Return successful node assignment;
    if x *is closer to* $nodes[id].initLoc$ *than* $nodes[GF[x].assignedNode].initLoc$ **then**
        Set $nodes[id].loc \leftarrow x$;
        Call *assignNode*($GF[x].assignedNode$);
        Return successful node assignment;
    Return unsuccessful assignment of node; // *id* is a *free* node

Algorithm

Algorithm 3: releaseNode()

Input: $unassignedGridPoints$ containing unassigned grid positions of GF and $freeNodeList$ containing unassigned nodes.

Output: Assigned $free$ node at the proper empty grid points.

while $unassignedGridPoints \neq \phi$ *and* $freeNodeList \neq \phi$ **do**

- **Remove** a grid point from $unassignedGridPoints$ say x;
- **Remove** a node, say id, from $freeNodeList$ closest to x;
- **Set** $nodes[id].loc \leftarrow x$ and $GF[x].assignedNode \leftarrow id$;

if $unassignedGridPoints = \phi$ **then**

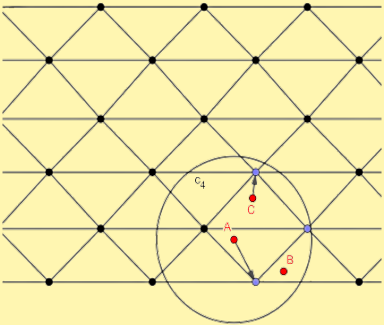
- **Report** all the nodes are in hexagonal configuration;

else

- **Report** Healing of entire hole is not possible;

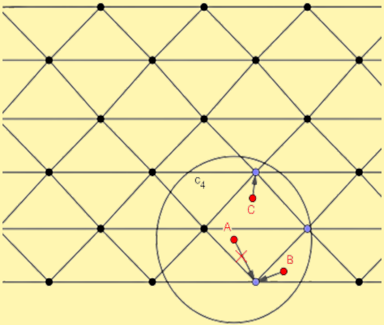
Algorithm

- The nodes A and C assign themselves to their nearest vacant grid point.



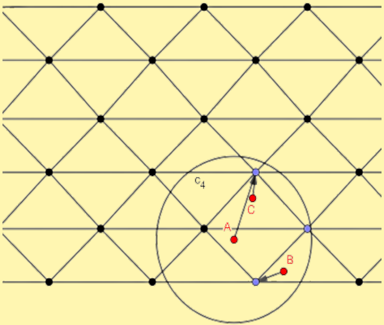
Algorithm

- B wants to assign itself to the grid A is already assigned.
- distance of A is greater than distance of B from the grid.
- B assigns itself there and A de assign itself



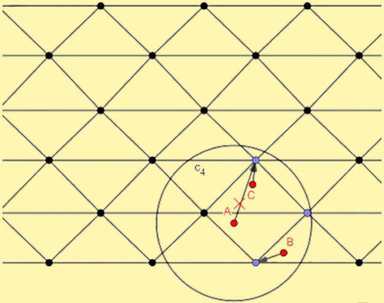
Algorithm

- *A* tries to assign itself to the next nearest grid point.



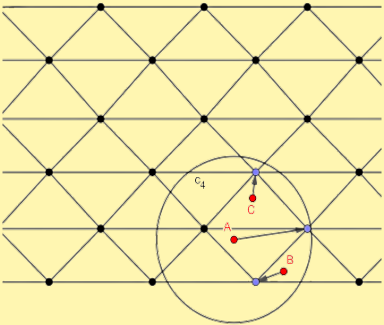
Algorithm

- C is already assigned there.
- distance of C from the grid point is lesser than distance of A from the grid point.
- A does not assign itself there.



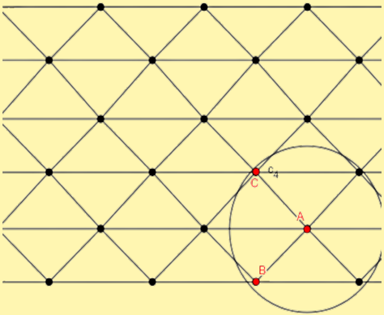
Algorithm

- *A* now assign itself to the next nearest and vacant grid point.



Algorithm

- A, B , C moves to their corresponding assigned grid points.



Performance Analysis

Results

from lemma 1 and lemma 2 the theorem follows:

Theorem 1

In a 6-neighbor grid with grid unit $l = \sqrt{3}s$, a $2s$-*disk* contains at least four grid points and at most seven grid points.

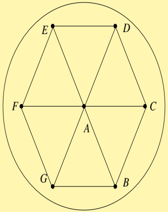


Figure: 7 grid points

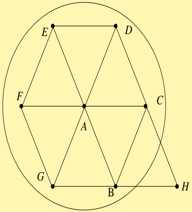


Figure: impossibility of grid points more than 7

Results

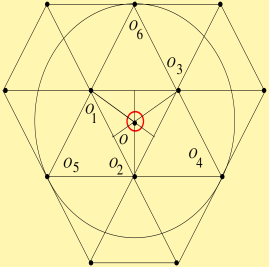


Figure: 6 grid points

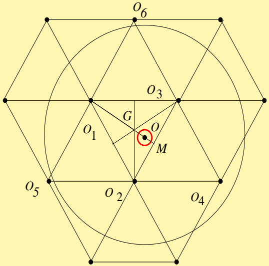


Figure: 4 grid points

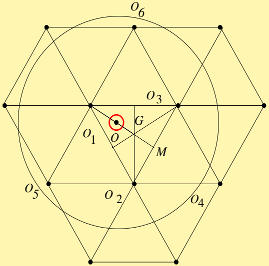


Figure: 5 grid points

Correctness and Complexity

- In Algorithm 2 the recursive call of the function *assignNode(id)* can be called at most 7 times for each node. So it terminates within $O(n)$ times.
- the movement is minimal.

References

Li, X. and Hunter, D. (2008).
Distributed coordinate-free hole recovery.
pages 189 – 194.

Mahboubi, H. and Aghdam, A. G. (2017).
Distributed deployment algorithms for coverage improvement in a network of wireless mobile sensors: Relocation by virtual force.
IEEE Transactions on Control of Network Systems, 4(4):736–748.

Sau, B. and Mukhopadhyaya, K. (2013).
Localizability of wireless sensor networks: Beyond wheel extension.
In *Symposium on Self-Stabilizing Systems*, pages 326–340. Springer.

Zhang, H. and Hou, J. C. (2005).
Maintaining sensing coverage and connectivity in large sensor networks.
Ad Hoc & Sensor Wireless Networks, 1(1-2):89–124.

Thank You!