Innovations Syst Softw Eng (2016) 12:227–238
DOI 10.1007/s11334-016-0277-7

S.I. : ICACNI 2015

Self-organized area coverage in wireless sensor networks by limited node mobility

Dibakar Saha[1] · **Nabanita Das**[1]

Received: 20 November 2015 / Accepted: 13 March 2016 / Published online: 15 April 2016

Abstract For wireless sensor networks, monitoring large inaccessible areas where deterministic node deployment is not possible, self-organized techniques are in demand to cover an area using optimal number of nodes. In this paper, given an initial random deployment of mobile sensor nodes, we propose a simple and novel technique for self-organized node movement to satisfy the coverage of the given region of interest using a least number of nodes, such that the maximum node displacement is minimized. We present a simple centralized algorithm and also a distributed version of it for node placement. Moreover, in case of a node failure, a distributed fault recovery algorithm is proposed to replace it locally utilizing the available free nodes. Analysis, simulation, and comparison studies show that the proposed algorithms with less neighborhood information result in significant improvement in terms of average and maximum displacement of a node, rounds of communication, and number of active nodes.

Keywords Area coverage · Node deployment · Sensing radius · Wireless sensor networks · Hexagonal tessellation

1 Introduction

In many applications of pervasive computing from home and health care to environment monitoring and intelligent transport systems, it is often required to place the sensors or computing nodes or access points to offer services over a predefined area. In wireless sensor networks (WSN), a large number of sensor nodes are spatially distributed over an area to collect ground data for various purposes such as habitat and ecosystem monitoring, weather forecasting, smart health-care technologies, precision agriculture, homeland security and surveillance. For all these applications, some active nodes should always cover the area to be monitored. Hence, for these networks, the coverage problem has emerged as an important issue to be investigated. So far, many authors have modeled the coverage problem in various ways, but most of them considered static nodes, where basically the region is overdeployed and a minimal set of nodes is selected to remain active to ensure coverage. In various forms, the problem is *NP-hard*, and it is more challenging to achieve even near-optimal solutions with limited amount of computation, communication and energy in the tiny sensor nodes.

In some cases, like mobile surveillance, mobile ad hoc networks, wireless sensor networks, etc., the nodes may have limited mobility, though displacement also needs some energy and should be kept minimum to save energy. To avoid message overhead for information gathering in a central node, these networks are preferred to be self-organized, so that nodes can take decision based on their local information only.

In this paper, given an initial random deployment of n mobile sensor nodes over a 2-D region, the area is dynamically tessellated by regular hexagons, and a few *target points* are identified to be filled up by an optimal number of nodes mutually exclusively such that the maximum node displacement is minimized. In WSNs, maximum node displacement is an important parameter. Since node movement exhausts energy, minimization of maximum node displacement helps to enhance the network lifetime. An $O(m^2 + n)$ centralized heuristic and a simple self-organized distributed algorithm with $O(d \cdot k^2)$ computation in each node are developed to

✉ Dibakar Saha
dibakar.saha10@gmail.com

Nabanita Das
ndas@isical.ac.in

[1] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

satisfy the coverage of a given region of interest, where m is the total number of target points, d is the maximum degree of a node, and k is an application parameter, a small integer, usually 1 or 2. After deployment, each node computes its nearby *target points* and respective distances. Next, a unique node is selected in a distributed fashion based on local position information only, to fill up each target point mutually exclusively, such that the maximum displacement of nodes is minimized to make the procedure energy efficient. The set of selected nodes is made active to cover the area. Compared to the works in [2], the computation involved in this algorithm is significantly simple; it requires no location information of its neighbors and converges faster in two rounds only. Simulation results show that the proposed method outperforms the earlier techniques in terms of number of nodes activated, computation and communication rounds, and finally the average and maximum displacement experienced by the nodes.

The rest of the paper is organized as follows. Section 2 presents the literature survey and our contribution. Section 3 defines the problem. Section 4 presents the movement-assisted centralized and distributed algorithms for self-deployment. Section 5 briefly describes the fault recovery technique. Section 6 evaluates the performance of the proposed algorithms by simulation. Finally, Sect. 7 concludes the paper.

2 Related work

For deterministic node deployment, centralized algorithms can be followed to maximize the area coverage assuming the area covered by each node to be circular, square, etc. [11,16,17]. Many authors solved the coverage problem by deterministic node placement techniques to maximize the network lifetime and to minimize the application-specific total cost. In paper [4], the authors investigated the node placement problem and formulated a constrained multivariable nonlinear programming problem to determine the locations of the nodes and data transmission pattern to optimize the network lifetime and the total power consumption.

The authors in [12,18] proposed random and coordinated coverage algorithms for large-scale WSNs. But unfortunately, in many potential working areas, such as in remote harsh environments, disaster affected regions, toxic regions, etc., sensor deployment cannot be done deterministically.

For random node deployment, virtual partitioning is often used to decompose the query region into square grid blocks and the coverage problem of each block is investigated [7,13,14]. But it is evident that whether the node deployment is deterministic or random, there is little scope of improving the coverage once the nodes are spatially distributed if they are static. Hence, mobility-assisted node deployment for efficient coverage has emerged as a more challenging problem.

Many approaches have been proposed so far, based on virtual force [6,22,23,27], swarm intelligence [8,10], and computational geometry [19], or some combination of the above approaches [5,15,20].

In [24], a movement-assisted node placement method is proposed based on van der Waal's force where the relationship of adjacency of nodes was established by Delaunay triangulation and the force is calculated to produce acceleration for nodes to move. However, the computation involved is complex and takes a large number of iterations to converge. The authors in [2] proposed a distributed algorithm for the autonomous deployment of mobile sensors called *push and pull*, where sensors coordinate their movements to achieve a complete and uniform coverage. In [19], based on Voronoi diagram, the authors designed and evaluated three distributed self-deployment algorithms for controlling the movement of sensors to achieve coverage. In these protocols, the sensors move iteratively, eventually reaching the final destination. These iterative procedures are computation intensive, may cause longer displacement of nodes, and may take longer time to converge as well. Moreover, each node requires the location information of its every neighbor to execute the algorithm demanding more message overhead, energy, and memory.

Our contribution In this paper, given a random node deployment over a 2-D region, we focus on a simple method to cover the region by a minimum number of sensor nodes such that the maximum displacement of nodes is minimized to save energy in individual nodes for longer life of the network. Here, we follow a hexagonal tessellation pattern and attempt to place a node at each vertex and the center of each hexagon termed here as *target points*. We first propose an $O(m^2 + n)$ centralized greedy heuristic, where m and n are the total number of *target points* and number of nodes, respectively, and $m \ll n$. In the centralized algorithm, we always try to fill a target point T_i by selecting a nearest node s_i such that the maximum displacement is minimized. Next, we propose a lightweight self-organized distributed algorithm with $O(d \cdot k^2)$ computation in each node, where d is the maximum degree of a node and k is a parameter. Since, by the proposed method, the area is minimally covered, fault tolerance or fault recovery is an important issue and is to be addressed with due care. In wireless sensor networks, nodes often fail due to low energy, hardware degradation, environmental changes, inaccurate readings, etc. If any *active* node fails to work, it results in an uncovered region and should be substituted immediately. To alleviate this problem, we propose a distributed fault recovery algorithm, by which idle nodes locally sense the unfilled target point in its neighborhood and execute a distributed algorithm to fill the *target point*. We compare our proposed centralized and distributed algorithms with the algorithm proposed in [2]. Comparison

studies show that the proposed algorithm with less neighborhood information results in significant improvement in terms of average and maximum displacement of a node, rounds of communication, and number of active nodes, respectively.

3 Proposed model and initialization

3.1 Problem overview

Let a set of n nodes $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be deployed randomly over a 2-D region A. It is assumed that nodes are homogeneous and cover a circular area with fixed sensing radius r. Now, the question is what is the minimum number of nodes to cover the given area A. To find the answer, we concentrate on the optimal placement of minimum number of nodes to cover the area, such that the maximum displacement of a node is minimized.

3.2 Area tessellation

To cover a given rectilinear area by homogeneous nodes, each with fixed sensing radius r, it is evident that if nodes can be placed deterministically at the target points as shown in Fig. 1, the area is fully covered using a minimum number of nodes. Here, the nodes are placed in such a way that the area is fully covered and the overlapped region is minimum. The positions of all the nodes basically defines a set of regular hexagons of side $\sqrt{3}r$ that tessellates the area as shown in Fig. 2.

Definition 1 The sensor nodes are placed exactly on the vertices and the centers (where the principal diagonals meet) of the regular hexagons, termed here as the *target points*, as shown in Fig. 1.

In [3], it is proved that such node placement technique maximizes the area coverage using a minimum number of nodes. In this case, the minimum number of nodes to be placed is the same as the total number of *target points* and

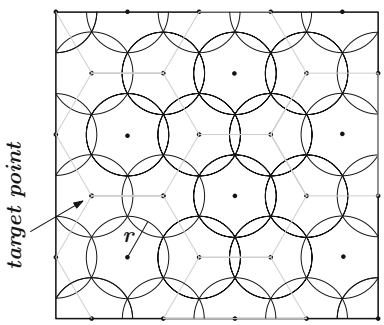

Fig. 1 Target points

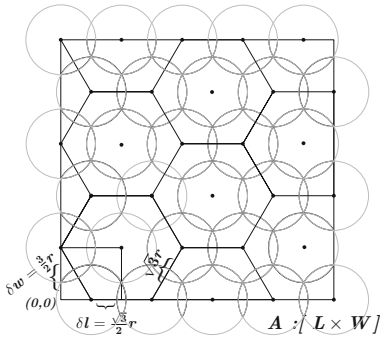

Fig. 2 Hexagons with target points over the given area A

can be computed easily as a function of the sensing radius r as shown below.

Let A be a 2-D axis-parallel rectangle $L \times W$ with $(0, 0)$ as the bottom-left corner point, termed here as the origin of area A, as shown in Fig. 2. For any arbitrary bottom-left corner point (x_0, y_0), the origin is to be translated appropriately. The area A is tessellated with regular hexagons of side $\sqrt{3}r$. It is to be noted that the *target points* lie along some rows and columns parallel to the x-axis and y-axis, respectively. Rows are separated by a distance:

$$\delta w = \sqrt{3r^2 - \frac{3r^2}{4}} = \frac{3}{2}r.$$

Similarly, columns are separated by a distance:

$$\delta l = \frac{\sqrt{3}}{2}r.$$

Hence, given an area A, and assuming that $L = p.\delta l$ and $W = q.\delta w$, where p and q are integers, from Fig. 2, it is clear that each even row-i starts with a target point $(0, i.\delta w)$, whereas each odd row-j starts with a target point $(\delta l, j.\delta w)$, $1 \leq i, j < q$. Hence given the area A, the total number of *target points* is

$$m = \left(\frac{2L}{\sqrt{3}r} + 1\right) \cdot \left(\frac{2W}{3r} + 1\right).$$

Therefore, to cover the area A, the number of nodes to be deployed is $n \geq m$ to fill up the *target points* exclusively. However, in practice, with random distribution of nodes, the area to be monitored is overdeployed, and $n \gg m$ providing sufficient redundant nodes to ensure coverage and fault tolerance.

Remark 1 It is interesting to see that the node distribution shown in Fig. 1 essentially covers larger area A': $(L+\sqrt{2}r) \times W$, as shown in Fig. 3, without hole.

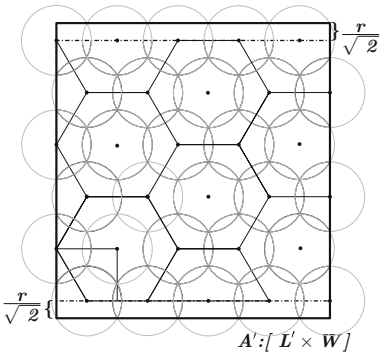

Fig. 3 Area extension

However, in this paper, the algorithms compute the position of the *target points* based on the origin only, without any dependence on L or W, as has been explained in the following section.

3.3 Nearest target point

Given the origin of a 2-D area A, the locations of the specific target points are to be determined so that appropriate nodes can move toward it for coverage. Distributed procedures are proposed to avoid the large message overhead associated with information gathering at a particular node. Let a set of n nodes $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be deployed randomly over the 2-D region A. Each node-i only have the information of its physical location (x_i, y_i) and the sensing range r. Now, to estimate the location of its nearest target point, it should have the knowledge of the origin, i.e., the bottom-left point of the area. The sink may directly broadcast it to all nodes when it is a static point. In case the area of interest is dynamic, or depends on the deployment, the nodes may determine the origin as the point with minimum abscissa and ordinate of all the nodes deployed as described below. Here, during initialization, each node-i broadcasts its own location (x_i, y_i) and maintains two variables initialized as $x_{\min} = x_i$ and $y_{\min} = y_i$, to keep the minimum abscissa and ordinate of all of the deployed nodes. It receives the messages with locations (x_j, y_j) from other nodes-j, and if $x_j \leq x_{\min}$ and/or $y_j \leq y_{\min}$ the values of $x_{\min}$ ($y_{\min}$) are changed appropriately, and if there is any update, the new value is broadcast again, otherwise it is ignored. In this way, after sufficient time, say T, all the nodes will converge to the same value of $x_{\min}$ and $y_{\min}$ and consider it as the origin of the area under consideration. In case of an event, the affected nodes may define the event area in terms of this origin dynamically. In the worst case, each node may have to transmit n messages to complete the procedure. To minimize the message communication overhead, instead of using all nodes, the boundary nodes only may perform the task. The boundary nodes compute the origin and broadcast in the network, so that all nodes may know the origin information and depending upon the origin it computes its *target point*. Note that the boundary nodes can be found by any of the techniques proposed in [9,25], or [21].

After the initialization phase, the nearest target point is to be computed by each node. We assume that each node knows the origin $(x_{\min}, y_{\min})$ of A. Next, each node i at location (x_i, y_i), attempts to find out its nearest target point. It computes

$$t_y(i) = \mathrm{NI}\left(\frac{y_i - y_{\min}}{\frac{3}{2}r}\right)$$

and

$$t_x(i) = \mathrm{NI}\left(\frac{x_i - x_{\min}}{\sqrt{3}r}\right) \quad \text{when } t_y(i) \text{ is even},$$
$$= \frac{|(x - x_{\min}) - \frac{\sqrt{3}}{2}r|}{\sqrt{3}r} \quad \text{otherwise}.$$

Here, $\mathrm{NI}(x)$ denotes the nearest integer value of x. Next, it finds the location of its nearest target point (x_{Ti}, y_{Ti}) as:

$$y_{Ti} = t_y(i) \cdot \frac{3}{2}r$$
$$x_{Ti} = t_x(i) \cdot \sqrt{3}r, \quad \text{when } t_y(i) \text{ is even},$$
$$= t_x(i) \cdot \sqrt{3}r + \frac{\sqrt{3}r}{2} \quad \text{otherwise}.$$

Remark 2 Once the origin of the region of interest is known, nodes can estimate their nearest target points independent of the dimensions of area A, i.e., L or W.

Lemma 1 *After deployment, each node may have one, two, or at most three nearest target points.*

Proof Given the origin of the area A, the positions of the target points T_i, and hence its circular covered area C_i has been fixed. Now, initially, a node s_i within a circle C_i around a target point T_i may lie either in the region (a) not overlapped by any adjacent circles, and hence will have just a single nearest target point T_i, or, (b) within the region overlapped by an adjacent circle C_j, around a target point T_j, and may be equidistant from both T_i and T_j, and thus have two nearest target points T_i, and T_j, or, (c) it may lie on the intersection point of three adjacent circles having three nearest *target points*, respectively. □

3.4 Region of k-influence

Ideally, any node may compete for any target point to cover. However, it increases the message overhead to co-ordinate

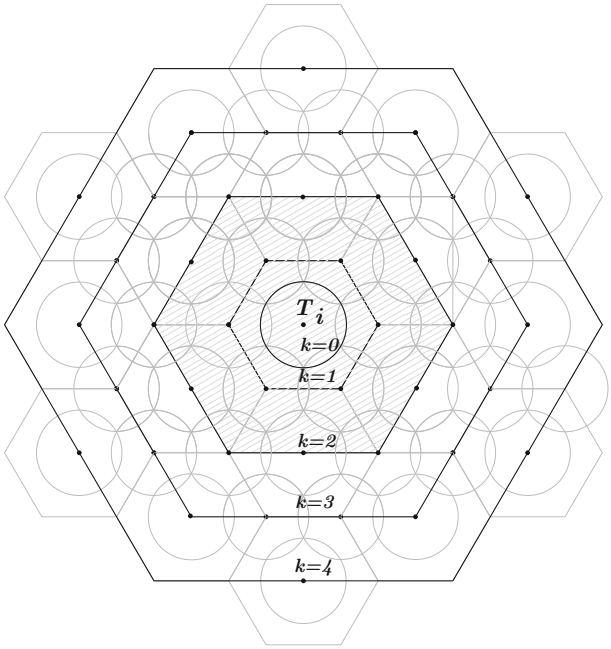

Fig. 4 Region of k-influence of target point T_i

among nodes that are far away and requires longer node displacement. Unless the network is very sparse, we may limit the influence of a target point within a region around it, without affecting the optimality of the solution.

Definition 2 Two target points are adjacent or one distance apart, if their circular covered areas are overlapping.

Definition 3 The target points at distance k from a target point T_i lie on a hexagon termed as ring-k of T_i, as shown in Fig. 4.

Definition 4 Region of k-influence of a target point T_i is defined as the area covered by ring-k of T_i, as shown in Fig. 4, for $k \geq 1$.

Remark 3 For $k = 0$, the region of k-influence of a target T_i is the circle C_i of radius r around T_i, i.e., the area covered by T_i only, as shown in Fig. 4.

Now, let us consider that for a target point T_i, only the nodes within its region of k-influence can compete, where k is a small integer. It is evident that depending on the node density, we may vary the value of k to achieve a satisfactory coverage. It is easy to see that the total number of target points within the region of k-influence of any target point T_i is given by:

$$
\begin{aligned}
N(k) &= 1 \quad \text{for } k = 0, \\
N(k) &= 1 + 6 + 12 + \cdots + (k-1) \cdot 6 \\
&= 3k^2 + 3k + 1, \quad \text{for } k \geq 1.
\end{aligned}
$$

Theorem 1 *If there exists at least $N(k)$ number of sensor nodes within the region of k-influence of each target point T_i, full coverage without any hole can be achieved with an upper bound of r for $k = 0$ and $k\sqrt{3}r$, for $k \geq 1$ on node displacement.*

Proof If within the region of k-influence of every target point T_i, there exists at least $N(k)$ number of sensor nodes, it is evident that each T_i may get at least one node to fill it up, since the region of k-influence of a target point T_i have exactly $N(k)$ number of target points. Now from Fig. 4, it is clear that to fill up the target point T_i, the maximum distance a node within the region of k-influence of a target point T_i may have to move is only r for $k = 0$, and $k\sqrt{3}r$ otherwise. Hence, the proof. □

3.5 Role of communication range

So far, we have mentioned the sensing range of the sensor node-i that defines the circular area with radius r, centered at node-i to be the area covered by node-i. When a node executes a distributed algorithm, after some computation, it co-ordinates with its neighbors by communication. Hence, it is very important to decide the influence of its action, or its neighborhood with which it can communicate directly. For that, we should specify the communication range r_c of a node-i which indicates that when a node-i transmits, a node-j can receive the packet if and only if the distance between the nodes is $d(i, j) \leq r_c$. It is important to note that for all practical purposes, communication range r_c is independent of the sensing range r, since r_c is determined by the transceiver hardware of node, and on the other hand, r is the property of the sensing hardware. However, both coverage and connectivity are equally important for the proper functioning of wireless sensor networks. It has been already proved that a node distribution satisfying the coverage constraint also guarantees connectedness if $r_c \geq 2r$ [26]. Here, it is evident that as r_c increases, a node may directly communicate with a larger number of neighbors to fill up a target point collaboratively, to result in a near optimal solution. In case of sparse networks, for better coverage, we may need to apply larger values of k for the region of k-influence to achieve better coverage. But larger value of k will demand either larger value of r_c, or more number of communication rounds to exchange messages. Here, we assume that for the given value of k, the value of r_c is sufficiently large, so that all the nodes within the region of k-influence of each T_i may communicate directly.

For example with $k = 1$, each node should cooperate with its neighbor nodes which are within a distance of $2\sqrt{3}r$, as shown in Fig. 5. Hence, to communicate with all these nodes directly, $r_c \geq 2\sqrt{3}r$.

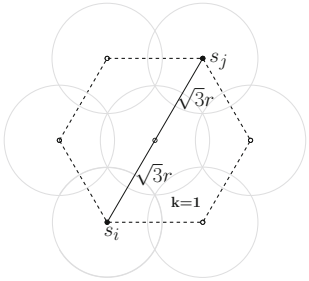

Fig. 5 Communication range between node s_i and s_j

4 Movement-assisted algorithms for area coverage

Given a random deployment of n nodes over a region with m target points, $n \gg m$, our objective is to place unique nodes to each target point minimally such that the maximum node displacement is minimized.

4.1 Centralized algorithm

To select unique node for each target, we consider a bipartite graph $G(\mathcal{T}, S, E)$, where $\mathcal{T}$, S, and E are the set of *target points* and sensor nodes, and a set of edges, respectively, and each edge $e(T_i, s_i)$ is associated with a weight $d(T_i, s_i)$, the Euclidean distance between node s_i, and target point T_i, if and only if $d(T_i, s_i) \leq k.\sqrt{3}r$ for a given k, as shown in Fig. 6.

It is evident that the problem of finding unique nodes for each *target point* is same as the classical problem of finding *maximum matching* for the bipartite graph G, to select a set of edges such that every vertex of the graph is incident to exactly one edge of the matching. This problem can be solved by [1] that runs in $O(|E| \cdot \sqrt{n})$ time in the worst case, where $|E|$ is the number of edges in the graph, and n is the number of vertices of the graph. However, our problem is even harder. Since over and above the maximum matching, we want to minimize the maximum displacement of a node, i.e., to keep the maximum edge weight minimum in the *matching*.

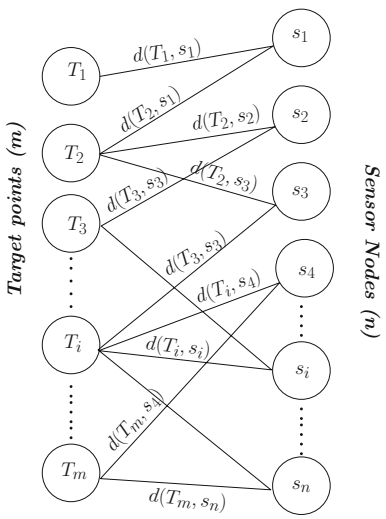

Fig. 6 Bipartite graph $G(\mathcal{T}, S, E)$

Here, we propose a centralized greedy heuristic to fill the *target points* assuming that the global network information is available to a central node. In this algorithm, each *target point* T_i and each node s_i are initialized by $node\_status(s_i) = 0$, and $target\_status(T_i) = 0$, to signify that s_i is free and T_i is unfilled. When a node s_i gets selected for a *target point* T_i, the status of both will be 1. Next, we make a list of nodes, $N[T_i]$ for each *target point* T_i. Here, all the nodes located within the region of k-influence of T_i are included in $N[T_i]$, sorted by Euclidean distance from T_i. Now for each *target point* T_i, we select a unique node s_i such that the Euclidean distance of s_i from T_i is the maximum among all other target points with the nearest node s_i. Then, T_i will be filled by the node s_i and s_i will be deleted from all $N[T_j]$ of the unfilled *target points*. This process is repeated until either $target\_status(T_j) = 1$ or $N[T_j] = \phi,\ \forall j, 1 \leq j \leq m$.

The details of the procedure *Algorithm 1* are presented below.

Algorithm 1: Centralized algorithm

```
Input: Target points : m; Sensor nodes: n; node_status; target_status; k;
Output: Filled target points;
for each node s_i do
    node_status[s_i] ← 0;
for each target point T_i do
    target_status[T_i] ← 0;
    flag[T_i] ← 0;
for each target point T_i do
    for each node s_i do
        if T_i lies within the region of k-influence then
            include node s_i in N[T_i] in sorted ascending order (by Euclidean
            distance d(T_i, s_i));
terminate ← false;
while terminate == false do
    for each target point T_i do
        if target_status[T_i] == 0 and N[T_i] ≠ {φ} then
            take first node s_i ∈ N[T_i];
            for each target point T_j do
                take first node s_j ∈ N[T_j];
                if s_i == s_j then
                    if d(s_i, T_i) < d(s_i, T_j) then
                        flag[T_i] ← 1;
        if flag(T_i == 0) then
            target_status[T_i] ← 1;
            node_status[s_i] ← 1;
        else
            Remove s_i from N[T_i];
    for each target point T_i do
        if target_status[T_i] == 1 then
            terminate ← true;
```

Correctness and complexity analysis By hexagonal tessellation, we place at least one node at each vertex and center of the hexagon termed as *target points*. Now for each *target point*, a unique node is selected for the placement by *Algorithm 1* that attempts to minimize the maximum possible node dis-

placement, by selecting a target point T_i nearest to a node s_i, but at maximum distance. There may exist a node s_j at less distance, but if s_j fills T_i, s_i will attempt to fill another *target point* T_j, where $d(T_j, s_i) \geq d(T_i, s_i)$, thereby increasing the maximum displacement of a node. When all the *target points* are filled, the algorithm is terminated. Note that if any *target point* T_i remains unfilled that means the total number of deployed sensor nodes is less than the total number of target points within the k-influence region of T_i. The number of active sensor nodes to cover the region is always bounded by the total number of *target points*.

In the centralized algorithm, the initialization phase for n nodes and m *target points* takes $O(n)$ and $O(m)$ time, respectively. Next, for each *target point* T_i, the process of making the list of nodes $N[T_i]$ takes $O(m.k^2 \log k)$ time, assuming a uniform node density and considering a region of k-influence. Selection of a suitable node for each *target point* takes $O(m^2)$ time. Therefore, the total computation time is $O(m^2 + n)$.

4.2 Distributed algorithm

In wireless sensor networks, since nodes have limited computing and communication capabilities, it is always better to adopt distributed algorithms where nodes may take decisions with simple computation based on their local information only to have a global solution.

Algorithm 2: *Target point* computation

Input: node location: (x, y), Sensing radius: r
Output: *target point*: (x_{Ti}, y_{Ti})
$\delta w \leftarrow \frac{3}{2}r$;
$h \leftarrow \sqrt{3}r$;
$t_y \leftarrow NI(\frac{y}{\delta w})$;
if t_y *is even number* **then**
  $t_x \leftarrow NI(\frac{x}{h})$;
  $x_{Ti} \leftarrow t_x \times h$;
else
  $t_x \leftarrow |x - \frac{h}{2}|$;
  $t_x \leftarrow NI(\frac{t_x}{h})$;
  $x_{Ti} \leftarrow (t_x \times h) + \frac{h}{2}$;
$y_{Ti} \leftarrow t_y \times \delta w$;

Here initially, each node i remains in an active state. It is assumed that each node i knows its location (x_i, y_i) and the origin $(x_{\min}, y_{\min})$ of the area to be covered and maintains a list $NL[i]$ of its neighbors.

In the first round, each node-i assumes a virtual tessellation of the area with hexagonal tiles and computes its nearest *target point* $T_i(x_{Ti}, y_{Ti})$ by *Algorithm 2*. Next, node i computes all *target points* within the region of k-influence of T_i, and includes all the *target points* in a list $TL[i]$, and sorts them according to the distance from it. Now, node-i takes the nearest *target point* $(x_{i_1}, y_{i_1}) \in TL[i]$ and broadcasts a *target* $((x_{i_1}, y_{i_1}), d_i)$ message, where d_i is its distance from the second nearest target point in $TL[i]$. Next, node-i waits till it receives *target* messages from all of its neighbors in $NL[i]$. Then, it checks only those *target* messages containing the same *target point*. If it finds that its d_i is maximum from all its neighbors, node-i broadcasts *selected* $(i, T_i(x, y))$ message and moves toward the *target point*. The case of tie may be resolved by node-id. If a node i receives a *selected* $(j, T_j(x, y))$ message from node j, then it removes j from $NL[i]$ and the *target point* $T_j(x, y)$ from $TL[i]$. This procedure is repeated until $TL[i]$ or $NL[i]$ is empty. *Algorithm 4*, presented below, describes the sequence of steps of the procedure.

Algorithm 3: Node selection for *target point*

Input: node: i
Output: movement: true or false
movement=true;
for *each neighbor node* $j \in NL[i]$ **do**
  if *receives* (x_{Tj}, y_{Tj}, d_j) *message* **then**
    if $(x_{Ti}, y_{Ti}) == (x_{Tj}, y_{Tj})$ *// same target point* **then**
      if $d_i < d_j$ **then**
        movement=false;
      if $d_i == d_j$ **then**
        if $i > j$ **then**
          movement=false;

Algorithm 4: Distributed algorithm

Input: node i, $NL[i]$, $movement = true$, k;
Output: Active or idle mode;
for *each node* i **do**
  Compute nearest *target point* (x_{Ti}, y_{Ti}) (call *Algorithm 2*) and compute all other *target points* within the region of k-influence. Include all the *target points* in $TL[i]$ sorted by distance $\mathcal{D}[i]$;
  for $TL[i] \neq \{\phi\}$ **do**
    Take the first point (x_{Ti}, y_{Ti}) and d_i from $TL[i]$ and $\mathcal{D}[i+1]$ respectively;
    Broadcasts $target((x_{Ti}, y_{Ti}), d_i)$ message;
    Wait and listen until receives all target message from the neighbors in $NL[i]$;
    Call *Algorithm 3*;
    if *movement==true* **then**
      Move toward *target point* (x_{Ti}, y_{Ti});
      Broadcasts $selected(i, (x_{Ti}, y_{Ti})$ message;
    if $STATUS(i) == 1$ **then**
      Goto active Mode;
      Free $TL[i]$ and $\mathcal{D}_i$;
      Terminate;
    else
      Remove *target point* (x_{Ti}, y_{Ti}) from $TL[i]$;
      if *receives* $selected(j, (x_{Tj}, y_{Tj}))$ *message* **then**
        Remove *target point* (x_{Tj}, y_{Tj}) from $TL[i]$;
        Remove j from $NL[i]$;
  Terminate and goto idle Mode;

4.3 Correctness and complexity analysis

From the outline of the distributed algorithm, it is evident that each node i first computes the nearest *target point* T_i and next it computes all other *target points* within the region of k-influence of T_i and keeps all the *target points* in its target list $TL[i]$ in sorted order (by distance). Next, each node s_i broadcasts its nearest *target point* T_i along with its distance d_i from the second nearest *target point*. If more than one node computes for the same *target point*, the node at

maximum distance d_i is selected, to minimize the maximum displacement. If T_j is filled by other neighbor nodes, it is removed from $TL[i]$. If a node i finds that $TL[i]$ is empty, it goes to idle mode, otherwise it attempts to fill the *target point* in $TL[i]$. Note that, in the worst case, each node may attempt $N(k)$ times for filling a *target point*. Therefore, the process takes at most $N(k)$, i.e., $O(k^2)$ rounds to terminate.

4.3.1 Time complexity

It is evident that *Algorithm 2* is computed in constant time. To make a decision for selecting a unique node for each *target point*, each node waits until it receives all the *target* messages from its d neighbors. Each node takes $O(d)$ time to get the maximum distance value from its d neighbors. Therefore, *Algorithm 3* executes in $O(d)$ time, and d is the maximum number of neighbors of a node. Finally, each node attempts to fill up only $N(k)$ *target points*. Therefore, the computation in each node (*Algorithm 4*) is $O(d \cdot k^2)$, where k is a small integer, 1 or 2.

4.3.2 Message complexity

In the distributed algorithm, each node broadcasts at most $O(k^2)$ *target* messages considering a region of k-influence, and only one *selected* message. Therefore, per node, at most $O(k^2)$ messages are needed to complete the procedure. It is to be noted that for overdeployed networks, the value of k is 1 or 2 only.

4.4 Example

Let us consider a set of nine nodes, $\{s_1, s_2, s_3, \ldots, s_9\}$ deployed over a 2-D region as shown in Fig. 7. Note that initially, circle C_1 with center t_1 and radius r is empty. Each node s_i, where $1 \leq i \leq 9$, computes the nearest *target point* and also all neighbor *target points* within the region of k-influence of T_i, and includes them in its target list $TL[n_i]$ in sorted order. For $k = 1$, nodes have the following target lists:

$TL[s_1]$: $\{t_7, t_4, t_3, t_6\}$;
$TL[s_2]$: $\{t_1, t_4, t_6, t_3\}$;
$TL[s_3]$: $\{t_6, t_4, t_1, t_3\}$;
$TL[s_4]$: $\{t_1, t_4, t_5, t_2\}$;
$TL[s_5]$: $\{t_1, t_2, t_5, t_7, t_6, t_3, t_4\}$;
$TL[s_6]$: $\{t_6, t_4, t_5, t_7\}$;
$TL[s_7]$: $\{t_1, t_4, t_2, t_5\}$;
$TL[s_8]$: $\{t_6, t_4, t_5, t_7\}$;
$TL[s_9]$: $\{t_4, t_2, t_7, t_5\}$.

We assume that all nodes within the region of k-influence are directly connected. In round one, each node broadcasts a *target* message and waits for the target messages from its neighbors.

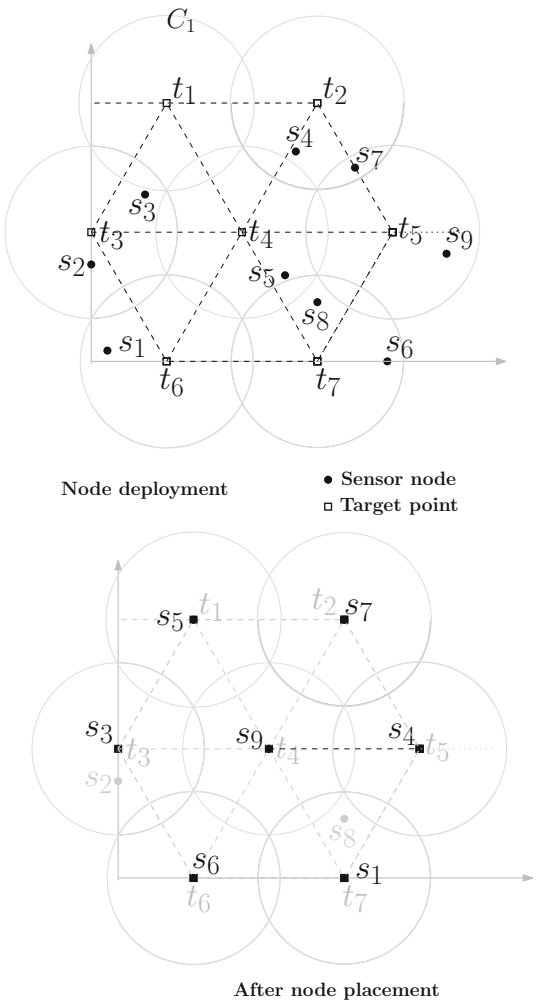

Fig. 7 Placement of nodes

In this example, node s_1 has the *target point* t_7, and there is no other claimant for the same target. Therefore, node s_1 moves toward t_7. On the other hand, nodes s_2, s_4, s_5 and s_7 report the same *target point* t_1, but the next target point distance for s_5 is the largest. So, node s_5 moves toward t_1. It is to be noted that after completion of round one, four *target points* are filled with nodes except the *target points* t_2, t_3 and t_5.

In the next round, idle nodes have the following target lists,

$TL[s_2]$: $\{t_3\}$;
$TL[s_3]$: $\{t_3\}$;
$TL[s_4]$: $\{t_5, t_2\}$.
$TL[s_7]$: $\{t_2, t_5\}$.
$TL[s_8]$: $\{t_5\}$.

Figure 7 shows the final placement.

5 Fault detection and recovery

In wireless sensor networks, a node may fail to work due to low energy, hardware degradation, inaccurate readings, environmental changes, etc. Since by the proposed algorithm, the area is minimally covered with minimum overlapping, if any *active* node fails to work, it results in an uncovered region and should be taken care of immediately. This paper focuses on the fault recovery problem in case of single or multiple node faults due to energy exhaustion only. To recover from such failures, it can be assumed that when the residual energy of a node reaches a threshold, it broadcasts a *recovery* message. To handle unexpected failures, each *idle* node periodically senses whether the target points within its region of k-influence are filled or not. If not, it broadcasts a *recovery* message. In either case, each *idle* node broadcasts a *target* message and waits for all target messages from its neighboring *idle* nodes and executes the algorithm defined in *Algorithm 3* to fill up the empty target point. *Algorithm 5* shows the brief outline of the process.

Algorithm 5: Fault recovery algorithm

Input: Node- i, $NL[i]$,$movement = true$
Output: Active or Idle modes
each Node i periodically sense whether its target point (x_{Ti}, y_{Ti}) is filled with an *active* node or not;
if (x_{Ti}, y_{Ti}) *is unfilled* **then**
  if *status of node i is active* **then**
    broadcast recovery(x_{Ti}, y_{Ti}) message;

if *a free node i receives an recovery*$((x_{Tj}, y_{Tj}))$ *message from node j* **then**
  Compute the distance d_i;
  broadcasts $target((x_{Ti}, y_{Ti}), d_i)$ message;
  Wait and listen until receives all target message from the neighbors $NL[i]$;
  Call *Algorithm 3*;
  if *movement==true* **then**
    Move toward target point (x_{Ti}, y_{Ti});
    broadcasts $selected(i, (x_{Ti}, y_{Ti})$ message;
    Goto active Mode;
  else
    Goto Idle Mode;
  Terminate;

Complexity In the first step, each node detects the unfilled target point in constant time. Next, the free node may execute *Algorithm 3* which takes $O(d \cdot k^2)$ time, where d is the maximum degree of a node. Therefore, the computation complexity in each node is $O(d \cdot k^2)$ considering a region of k-influence. For overdeployed networks, k is a small integer.

6 Simulation results

In our simulation study, we assume that n nodes, $50 \leq n \leq 400$, are distributed randomly over a 500×500 square unit area with radius $r = 28.86$ unit, such that the side of the hexagon is 50 unit. We consider the region of k-influence with $k = 2$ only, and it is assumed that r_c is sufficiently large such that all nodes within the region of 2-influence are directly connected. By simulation, the proposed algorithms are evaluated in terms of coverage, rounds of computation needed, and displacement of nodes. The graphs show the average value of 20 runs for 20 independent random deployments of nodes.

For typical instance, Figs. 8 and 9 show the displacement of nodes by the centralized and the distributed algorithms, respectively. It clearly shows that centralized algorithm performs better in terms of average node displacement, as is expected. However, the maximum node displacement that we attempt to minimize is comparable in both.

Figure 10 shows the variation in the number of computation rounds with n, the total number of nodes. With random node deployment, if no circular region C_i with center at a target T_i and radius r is empty, the procedure completes in a single round only. This fact is exactly revealed in Fig. 10. For $n = 100, 150, 200$, *target points* are filled up in two rounds, whereas for $n > 200$, the proposed technique takes a single round only to complete. Figure 11 shows the variation of coverage percentage with n, the total number of nodes deployed. With 105 *target points*, for $n = 50, 100, 150$ the coverage percentage is found to be 47, 84.57, and 99.36 % respectively. It gives an idea that how much an area should be overdeployed to achieve 100 % coverage. To compare the performance with [2], Fig. 12 shows that our proposed method always terminates in one or two rounds for an overdeployed random deployment, whereas the proposed technique in [2] takes at least four more number of rounds to complete

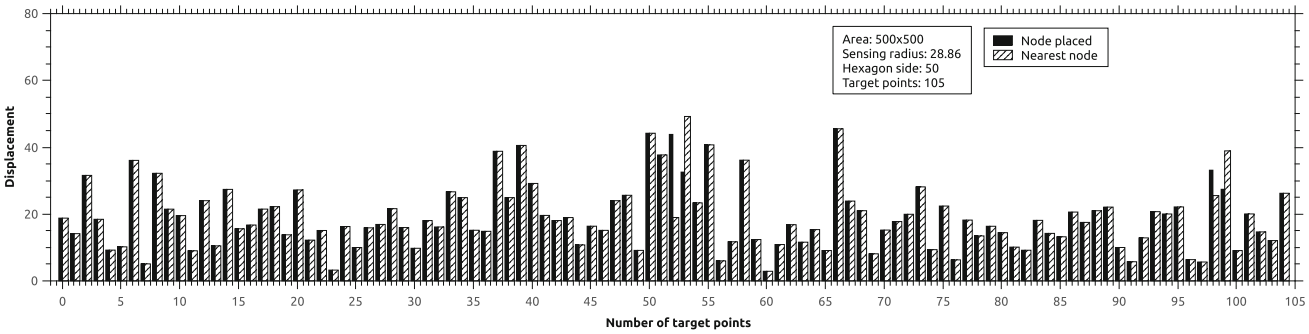

Fig. 8 Node displacement by centralized algorithm in comparison with the distance of the nearest node

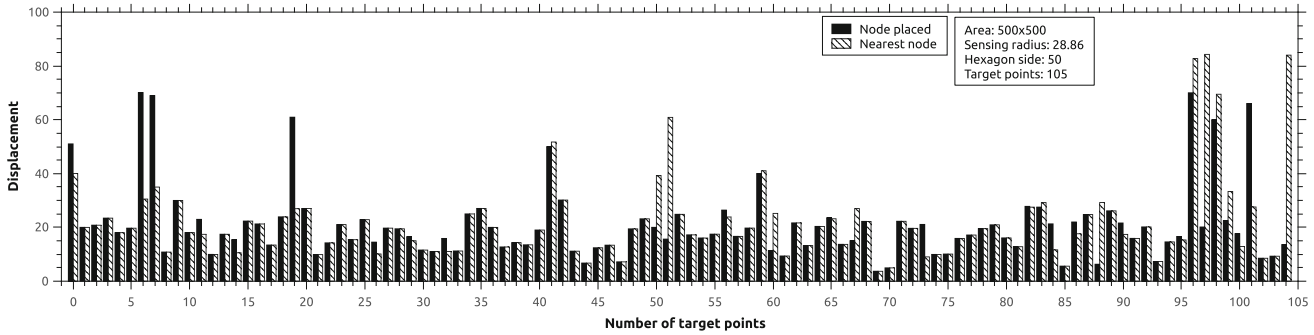

Fig. 9 Node displacement by distributed algorithm in comparison with the distance of the nearest node

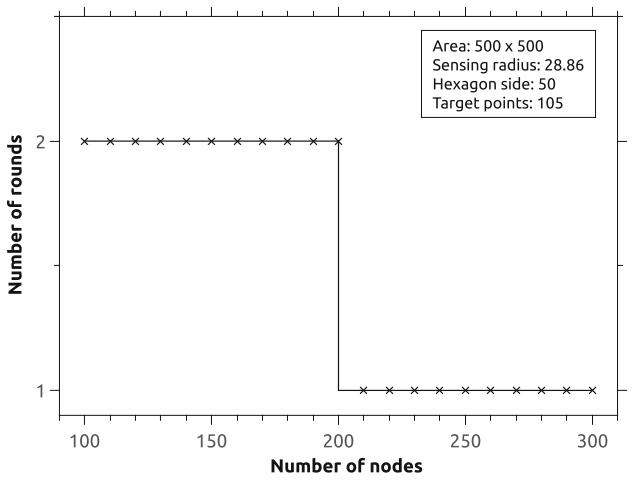

Fig. 10 Number of rounds for filling the *target points*

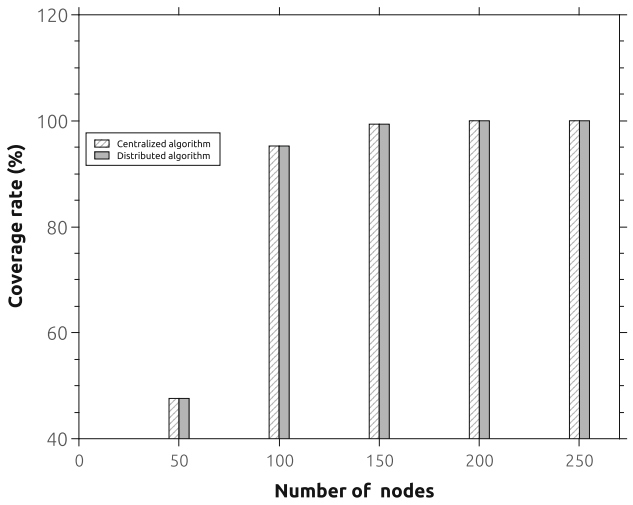

Fig. 11 Coverage rate with the number of nodes

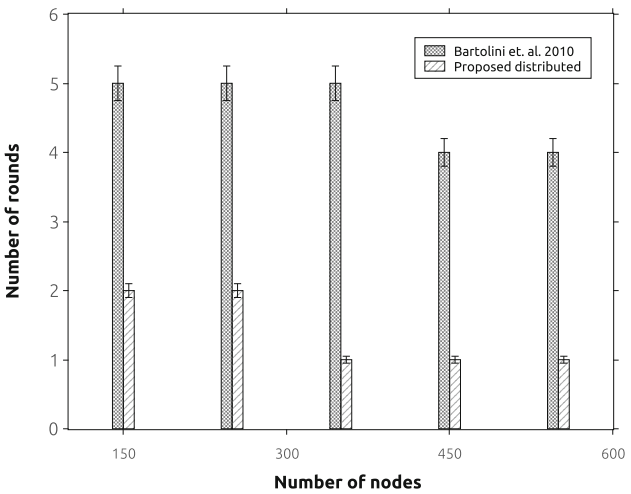

Fig. 12 Comparison of computational rounds with [2]

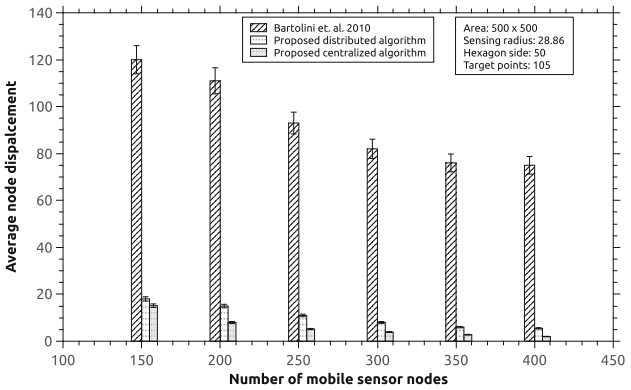

Fig. 13 Comparison of average node displacement with [2]

their procedure. For instance, with $n = 150$, and $r = 28.86$ unit, Fig. 13 shows the distances traversed by each node to fill up a target point.

In [2], the displacement due to pull or push of a node is much greater as shown in Fig. 13. Also, the proposed algorithms reduce the maximum displacement of a node by more than 200 %, as shown in Fig.14.

7 Conclusion

In this paper, we propose a self-organized node placement algorithm to cover a given region of interest in wireless sen-

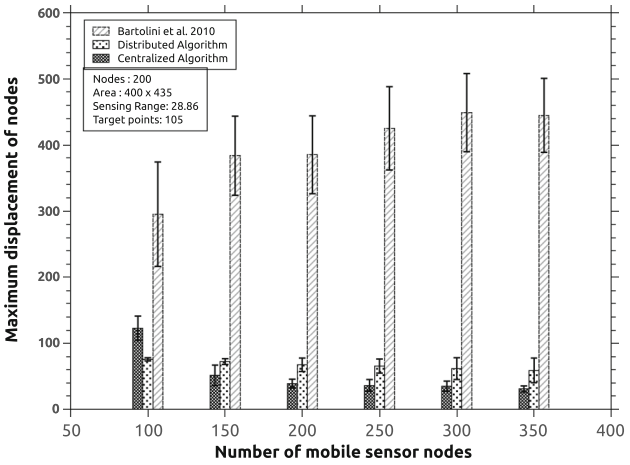

Fig. 14 Comparison on maximum node displacement with [2]

sor networks. The area is logically tessellated by regular hexagonal tiles starting from an origin. To get full coverage with random deployment of n nodes over a $2D$ region, we need to place unique nodes on every *target point*, which are essentially the vertices and the centers of the hexagons. The objective is to place a unique node to each target point, such that the maximum displacement of a node is minimized to enhance the network lifetime. We develop an $O(m^2 + n)$ centralized algorithm, where n and m are the total number of nodes and *target points*, respectively. With the knowledge of its own location and the origin, each node executes a simple self-organized distributed algorithm with $O(d \cdot k^2)$ computation complexity (d is the maximum number of neighbors of a node) and $O(k^2)$ message complexity considering a region of k-influence, to fill up all the *target points* mutually exclusively to minimize the maximum displacement. In case of failure of nodes, existing free nodes may take necessary action to fill up the empty *target points* to make the system fault tolerant. We evaluate the performance of our proposed model by simulation. It shows that the proposed algorithm with less neighborhood information results in significant improvement in terms of average and maximum displacement of a node, rounds of communication, and number of active nodes.

References

1. Alt H, Blum N, Mehlhorn K, Paul M (1991) Computing a maximum cardinality matching in a bipartite graph in time. Inf Process Lett 37(4):237–240
2. Bartolini N, Calamoneri T, Fusco E, Massini A, Silvestri S (2010) Push and pull: autonomous deployment of mobile sensors for a complete coverage. Wirel Netw 16(3):607–625
3. Brass P (2007) Bounds on coverage and target detection capabilities for models of networks of mobile sensors. ACM Trans Sens Netw (TOSN) 3(2):9
4. Cheng P, Chuah CN, Liu X (2004) Energy-aware node placement inwireless sensor networks. In: Global telecommunications conference, GLOBECOM, vol 5. IEEE, pp 3210–3214
5. Han YH, hwan Kim Y, Kim W, Jeong YS (2012) An energy-efficient self-deployment with the centroid-directed virtual force in mobile sensor networks. Simulation 88(10):1152–1165
6. Heo N, Varshney PK (2003) An intelligent deployment and clusteringalgorithm for a distributed mobile sensor network. In: IEEE international conference on systems, man and cybernetics, vol 5, pp 4576–4581
7. Ke W, Liu B, Tsai M (2011) The critical-square-grid coverage problem in wireless sensor networks is NP-complete. J Comput Netw 55(9):2209–2220
8. Kukunuru N, Thella BR, Davuluri RL (2010) Sensor deployment using particle swarm optimization. Int J Eng Sci Technol 2(10):5395–5401
9. Li X, He S, Chen J, Liang X, Lu R, Shen S (2011) Coordinate-free distributed algorithm for boundary detection in wireless sensor networks. In: Global telecommunications conference (IEEE GLOBECOM), pp 1–5
10. Liao WH, Kao Y, Li YS (2011) A sensor deployment approach using glowworm swarm optimization algorithm in wireless sensor networks. Expert Syst Appl 38(10):12180–12188
11. Luo CJ, Tang B, Zhou MT, Cao Z (2010) Analysis of the wireless sensor networks efficient coverage. In: Proc. of international conference on apperceiving computing and intelligence analysis (ICACIA), pp 194–197
12. Poe WY, Schmitt JB (2009) Node deployment in large wireless sensor networks: coverage, energy consumption, and worst-case delay. In: Conference on Asian internet engineering. ACM, New York, pp 77–84
13. Saha D, Das N (2013) Distributed area coverage by connected set cover partitioning in wireless sensor networks. In: Proc. of first international workshop on sustainable monitoring through cyber-physical systems (SuMo-CPS), ICDCN
14. Saha D, Das N (2013) A fast fault tolerant partitioning algorithm-for wireless sensor networks. In: Third international conference on advances in computing and information technology (ACITY). CSIT, India, pp 227–237
15. Saha D, Das N (2016) Self-organized node placement for area coverage in pervasive computing networks. In: Proceedings of 3rd international conference on advanced computing, networking and informatics, vol 43. Springer, New York, pp 365–376
16. Saha D, Das N, Bhattacharya BB (2014) Fast estimation of coverage area in a pervasive computing environment. In: Advanced computing, networking and informatics—volume 2, vol 28, pp 19–27
17. Saha D, Das N, Pal S (2014) A digital-geometric approach for computing area coverage in wireless sensor networks. In: Proc. of 10th international conference on distributed computing and internet technologies (ICDCIT), pp 134–145
18. Sheu JP, Yu CH, Tu SC (2005) A distributed protocol for query execution in sensor networks. In: Proc. of IEEE wireless communications and networking conference, vol 3, pp 1824–1829
19. Wang G, Cao G, Porta TL (2006) Movement-assisted sensor deployment. IEEE Trans Mob Comput 5(6):640–652
20. Wang X, Wang S, Ma JJ (2007) An improved co-evolutionary particle swarm optimization for wireless sensor networks with dynamic deployment. Sensors 7(3):354–370
21. Wang Y, Gao J, Mitchell JS (2006) Boundary recognition in sensor networks by topological methods. In: Proceedings of the 12th annual international conference on mobile computing and networking. ACM, New York, pp 122–133
22. Yu X, Huang W, Lan J, Qian X (2012) A novel virtual force approach for node deployment in wireless sensor network. In: 8th

international conference on distributed computing in sensor systems (DCOSS). IEEE, pp 359–363
23. Yu X, Huang W, Lan J, Qian" X (2012) A van der waals force-like node deployment algorithm for wireless sensor network. In: 8th international conference on mobile ad-hoc and sensor networks (MSN). IEEE, pp 191–194
24. Yu X, Liu N, Huang W, Qian X, Zhang T (2013) A node deployment algorithm based on van der waals force in wireless sensor networks. Int J Distrib Sens Netw 1–8
25. Zhang C, Zhang Y, Fang Y (2009) Localized algorithms for coverage boundary detection in wireless sensor networks. Wirel Netw 15(1):3–20
26. Zhang H, Hou JC (2005) Maintaining sensing coverage and connectivity in large sensor networks. Ad Hoc Sens Wirel Netw 1(1–2):89–124
27. Zou Y, Chakrabarty K (2003) Sensor deployment and target localization based on virtual forces. In: 22nd annual joint conference of the ieee computer and communications (INFOCOM), vol 2, pp 1293–1303