Autonomous Deployment in Mobile Sensor Systems

Hojin Ghim[1], Dongwook Kim[1] and Namgi Kim[2]
[1] Dept. of EECS, KAIST
373-1 Guseong, Yuseong, Daejeon, 305-701 - Korea
[e-mail: {hojin, kimdw}@nslab.kaist.edu]
[2] Dept. Of CS, Kyonggi University
San 94-6, Iui, Yeongtong, Suwon, Gyeonggi, 443-760 - Korea
[e-mail: ngkim@kyonggi.ac.kr]
*Corresponding author: Namgi Kim

Received March 26, 2013; revised June 10, 2013; revised July 23, 2013; accepted August 18, 2013; published September 30, 2013

Abstract

In order to reduce the distribution cost of sensor nodes, a mobile sensor deployment has been proposed. The mobile sensor deployment can be solved by finding the optimal layout and planning the movement of sensor nodes with minimum energy consumption. However, previous studies have not sufficiently addressed these issues with an efficient way. Therefore, we propose a new deployment approach satisfying these features, namely a tree-based approach. In the tree-based approach, we propose three matching schemes. These matching schemes match each sensor node to a vertex in a rake tree, which can be trivially transformed to the target layout. In our experiments, the tree-based approach successfully deploys the sensor nodes in the optimal layout and consumes less energy than previous works.

Keywords: Sensor deployment, mobile sensors, self-organization, sensor system, rake tree, layout model, regular triangular tessellation

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2012R1A1A1002133)

http://dx.doi.org/10.3837/tiis.2013.09.006

1. Introduction

The power of sensor networks comes from the distribution of small sensor nodes across a wide area with maximizing sensing coverage while preserving network connectivity. Many previous researches on sensor deployment assume that sensor nodes are randomly distributed in the entire target area before the deployment. Such uniformly random distribution of the sensor nodes may be realized manually. However, manually deploying a large number of sensor nodes across a wide area is quite costly or impossible due to hard-to-access terrain or a hostile environment. The nodes may be thrown away from aircraft. But this way makes it difficult to lay the nodes on the designated location and to support connectivity among the sensor nodes. Therefore, to reduce the cost of uniform distribution of the sensor nodes and prevent network partitioning, researchers have proposed self-deployment of mobile sensors [1].

However, the mobile sensor deployment still has challenges in target layout problem and sensor matching problem. The target layout problem is finding how to determine the optimal layout according to the current layout of the sensor nodes. A target layout should be optimal. It means that the target layout of the sensor nodes should cover maximum area. Also, in the target layout, each sensor node should be directly connected as many as possible for robustness. The sensor matching problem is finding a scheme that provides a way to match each sensor node to a point in the target layout. Once the sensor nodes move to the positions of the matched points, the target layout is constructed.

To solve these deployment problems, the greedy approach based on the concept of virtual force (or potential fields) among sensor nodes was proposed [2]. The greedy approach is very simple and flexible. Therefore, many deployment schemes have emerged from this approach [3][5][6][7][8]. However, the greedy approach fundamentally has shortcomings. The main cause of these shortcomings is that fact that it makes use of only local and neighbor information for deciding the movement of sensor nodes. In order to eliminate these shortcomings, we propose a new approach, called *tree-based approach*. The proposed approach uses the global information from the initial layout of sensor nodes. Using the global in formation, we decide the final destination for each node first and move accordingly one time only. Therefore, we can achieve the minimum energy consumption by following the shortest way.

In the following sections, we describe the previous works for deployment problem of the mobile sensor networks. Then, we describe the tree-based approach for deployment problem in detail in Section 3. Layout models that are mathematical models for target layouts are explained in Section 4. Matching schemes that can be adopted with the tree-based approach are described in Section 5. The performance is evaluated in Section 6 and conclusion is made in Section 7.

2. Related Work

A large amount of research literature exists for the deployment problem of mobile sensor networks. These can be categorized into three groups: the greedy approach, the coordinated approach, and the centralized approach.

In the greedy approach, deployment methods attempt to improve the coverage iteratively. In an iteration, each node decides and moves its target location based on various factors such as virtual forces from neighbors [2][5][6][7][8], the Voronoi cell associated to the node [3][4], holes around, and obstacles [6]. Some of them [5][7] try to make the regular triangular tessellation out of the iterative exertion of virtual forces. The iteration stops when the amount of movement in one iteration is below a predefined threshold or the maximum number of iterations has been reached. Minimax [3] exploits the fact that any point in a Voronoi cell has one sensor node as the closest node. Each node calculates the minimum enclosing circle of its Voronoi cell and moves to the center of the circle. In this way, it tries to cover all the points in the Voronoi cell and leave minimum holes in the cell. In ATRI [7], each sensor node exerts virtual force on its neighbors, letting them move toward a vertex of a hexagon. This exertion accelerates the construction of regular triangular tessellation even though it is not guaranteed. The performance of the greedy approach is highly dependent on the factors that decide the next location of the nodes. According to the factors, the methods using greedy approach tend to get aggressive or conservative in expanding the sensor networks at the boundary area. If they are too aggressive, the boundary will expand rapidly. The nodes at the boundary may lose connection with the neighbors and sometimes be forced to go backwards. On the contrary, if they are too conservative, the boundary expands too slowly so that they take too many iterations to cover the entire area. But, it is hard to balance the aggressiveness and conservativeness in the greedy approach because there is no central control and the nodes do not have the information about the surroundings outside their range.

The coordinated approach exchanges messages among sensor nodes and decides their target positions based on the exchanged information. To get the appropriate target positions, the sensor nodes should build some mechanism by themselves so that they can agree with each other about decisions. However, organizing this mechanism takes too much cost and energy unless careful consideration is taken. In the mechanism, the sensor nodes may not move at the same time. Some of the nodes may get their target positions after some of the others have moved. [9] assume a grid-like partition in the target area. A leader is elected in each cell and the leaders coordinate with each other to move nodes from cells with too many nodes to cells with too few nodes. [10] chooses a seed node among the nodes at first. Then, the neighbors of the seed move to the neighboring position of the seed in the regular triangular tessellation. This process propagates until all nodes are in the position of the regular triangular tessellation. However, the methods described above assume that the sensor nodes are dispersed throughout the target area initially. But, they will take significant amount of cost and energy if the initial deployment is concentrated in a small area.

Methods in the centralized approach are run at a separate server or the sink. The server or sink runs an algorithm to decide the final position of each sensor node and broadcast them. In the centralized approach, there are various methods based on virtual forces [2][11], circle packing [12], and one of the generic optimization techniques [13][14][15][16]. Obviously, however, these methods may be not affordable for some applications. The existence of a central server and the connections of the server and sensor nodes may be often not affordable due to application requirement or cost.

3. Tree-based Approach

In this paper, we assume that the sensor networks are composed of homogeneous sensor nodes. Each sensor node is capable of monitoring events or phenomena within a sensing range, R_S. The sensor nodes can communicate with other sensor nodes within a communication range, R_C. In addition, each sensor node detects its location by a localization device or a localization process. Once its location is detected, it sends a message that contains the ID and its own location before the initiation of deployment. So, other sensor nodes within its communication range can hear it. Lastly, the initial layout of the sensor nodes should not be partitioned.

With these assumptions, we are presenting a new approach for the deployment of mobile sensor networks, namely the *tree-based approach*. In the tree-based approach, a spanning tree is built over the initial sensor network to be used as a *base tree* at first. In order to build the base tree and elect the root node, we use the minimum diameter spanning tree (MDST) algorithm [17]. After that, the tree-based approach gathers global information about the initial layout and the base tree to the root node. Then, the target layout is determined at the root node by realizing a layout model. Thereafter, in the matching phase, information describing the target layout is disseminated to each sensor node following the hierarchy of the base tree. At the same time, each sensor node is matched to a vertex of the target layout according to a *matching scheme*. In the last movement phase, each sensor node moves to the matched vertex to finally create the target layout.

The tree-based approach provides several advantages. The tree-based approach can achieve the target layout exactly as specified by a layout model. Besides, the sensor nodes do not move more than needed to arrive at the target positions. Therefore, wasting of energy is avoided. Moreover, no priori information about the target area or the environment is required. Lastly, the duration of the deployment is predictable and the initial layout does not affect the performance of the deployment, whether the sensor nodes are evenly distributed or skewed.

The tree-based approach is self-organizable [18][19]. It uses the global informations which are collected and aggregated from the sensor nodes. The global informations that can be collected without incurring extra overhead of communication messages or additional delay include: the number of nodes and its descendants, and the geometric average of positions of each node and all its descendants in the base tree. By using and aggregating these informations only, the cost of tree-based approach can be limited with $O(n)$ number of messages within duration of $O(R)$ where n is the total number of sensor nodes and R is the radius of the base tree. As a result, the tree-based approach can be scalable in proportion to the number of sensor nodes.

4. Layout Model

A *target layout* for sensor deployment is a layout of sensor nodes that the deployment scheme is trying to achieve. A target layout with lots of vertices may take lots of space in the constrained memory of the sensor nodes. It may also require a large amount of energy to be transmitted between the sensor nodes in the process of deployment. Therefore, we need a scheme to represent a target layout within a limited amount of memory and energy. To that end, we propose a *layout model*. A layout model is a geometrical model that can produce a number of vertices when realized with a constant number of variables.

The tree-based approach for deployment can adopt various layout models according to the applications of the sensor networks. In this paper, we propose a layout model which achieves 1-coverage and k-connectivity ($k \leq 6$) with $R_C \geq \sqrt{3} \cdot R_S$ in free space. 1-coverage means that any point in the target area is covered with at least one sensor node. k-connectivity means that any sensor node is directly connected to at least k neighbor nodes for robustness. Other types of layout models also may be proposed to meet various requirements of sensor networks. But, due to the limits of space, we omit them.

In a sensor network with 1-coverage, k-connectivity ($k \leq 6$), and $R_C \geq \sqrt{3} \cdot R_S$, the optimal layout should cover the maximum area with a given number of sensor nodes without incurring any hole. Also, each sensor node should be directly connected with at least *k* neighbors. The pattern of the optimal layout with such properties is well-studied and known to be the regular triangular tessellation [7][20][21][22]. The regular triangular tessellation lets each sensor node have six neighbor nodes, which are equally distant from each other and delivers 6-connectivity. Each sensor node and its six neighbors form six regular triangles of which the length of each side is min(R_C, $\sqrt{3} \cdot R_S$) so that the coverage is maximized without holes.

Although the regular triangular tessellation guarantees the maximum coverage and 6-connectivity, it only specifies the pattern of the nodes without considering the shape of the boundary. The shape of the boundary of the target layout is supposed to be specified in consideration of the objectives of the sensor network and the terrain of the target area. Commonly, the *k*-connectivity requirement is relaxed for the sensor nodes at the boundary of the layout. Accordingly, in order to provide a optimal layout model that embeds the pattern of regular triangular tessellation and confines the practical boundary of the layout, we define a *hexagonal-boundary regular triangular tessellation* (HRTT) layout model.

Definition *hexagonal-boundary regular triangular tessellation* (HRTT) is a layout model in which all vertices constitute the regular triangular tessellation, with the boundary of the layout shaped as a hexagon. HRTT should satisfy the following conditions:

1. The center-distance of each boundary vertex is either the same as the radius of the layout or less than the radius by 1.
2. The boundary vertices of which the center-distance is the radius (called *farthest vertices*) are adjacent.

Center-distance denotes the hop-distance from a vertex to the center vertex, which is the graphic center of the layout. Also, *radius* denotes the maximum center-distance among all the vertices.

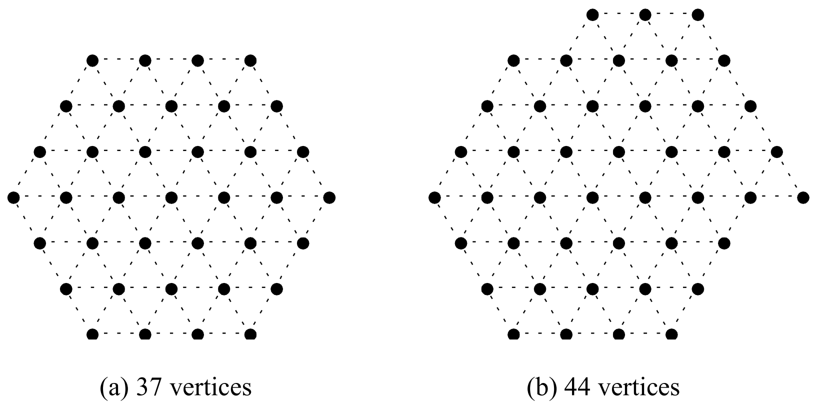


Fig. 1. An example of HRTT

Examples of HRTT with 37 and 44 vertices are shown in **Fig. 1**. The *Condition 2* of HRTT is followed by that the boundary vertices of which the center-distance is the radius – 1 are also adjacent because the boundary vertices of HRTT form a cycle. An instance of HRTT can be denoted by ((x_C, y_C), D, α, V) where (x_C, y_C) is the position of the center vertex; D is the distance between neighboring vertices; α is the angle of the first farthest vertex; and V is the number of vertices. For example, **Fig. 1a** shows a realization of HRTT, ((0,0), 1, 0, 37). In this case, all the boundary vertices are the farthest vertices with center-distance 3, which is the same as the radius. Given an HRTT, the exact positions of V vertices can be determined and matched to sensor nodes in order to be used as target positions.

In the tree-based approach, the root node can determine the target layout by realizing HRTT with $((x_{root}, y_{root}), \min(R_C, R_C \geq \sqrt{3} \cdot R_S), \alpha_1, V_1)$ where (x_{root}, y_{root}) is the position of the root node; α_1 is the angle of the first child of the root node; and V_1 is the number of sensor nodes. Therefore, in realizing HRTT, the root node needs only the total number of sensor nodes to be collected in the information-collecting phase, whereas other items of information are already known from itself or its neighbors.

5. Matching Scheme

The matching of sensor nodes to vertices in the target layout is equivalent to the minimum-weight connected matching problem [23]. However, an efficient distributed algorithm to solve this problem has not yet been discovered. Thus, we provide various heuristics for the problem, which are called *matching schemes*.

In this paper, three matching schemes for the target layout constructed with the HRTT layout model are presented. The proposed matching schemes employ the divide-and-conquer strategy. They divide the target layout such that each of the parts is matched to a group of sensor nodes. Also, these parts exploit the hierarchy of the base tree to group the sensor nodes recursively. Therefore, each group is composed of a child of the current node and the descendants of that child. Then, a part of the target layout, which contains the same number of vertices as the number of sensor nodes in the group, is matched to the group. Eventually, each sensor node can be matched to one of the vertices in the target layout.

5.1 Rake Tree Scheme

Prior to matching each node to a vertex of the target layout, the rake tree scheme builds a *rake tree*. (The concept of rake tree was introduced and the tree-based approach with the rake tree scheme was briefly proposed in [24]. However, we just introduced a simplified tree-based approach with the rake tree scheme only in that paper. We did not fully propose and evaluate the rake tree scheme. Moreover, we did not extend the tree-based approach with the balanced rake tree scheme nor the rake-ID space scheme at all.) A rake tree is a special type of spanning tree, as shown in **Fig. 2**. It can be transformed from the base tree with only reestablishing the relationships of the sensor nodes. A rake tree also can be easily transformed to an HRTT because each vertex of a rake tree is equivalent to a vertex in HRTT.

Using a rake tree, instead of HRTT, has several advantages for matching sensor nodes to vertices as follows. Firstly, it is easy to identify vertices by labeling them with unique IDs. Secondly, vertices can be easily grouped and assigned to a group of sensor nodes. Lastly, groups of vertices can be represented in a limited number of variables without enumerating every vertex in the group. These advantages are essential for the rake tree scheme to be scalable by limiting the size of communication messages to be constant.

5.1.1 Rake Tree

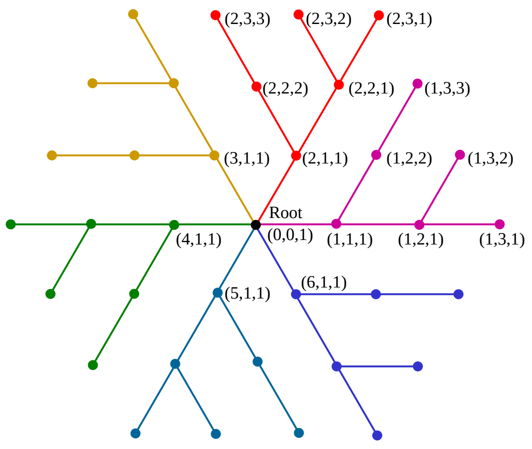


Fig. 2. A rake tree and rake-IDs

Definition *rake tree* is a spanning tree with the following properties:

1. Children of each vertex are ordered by the angles around their parent.
2. Vertices in the same level of each subsidiary are ordered by the ordinals of their parents and then by the ordinals of themselves.
3. The root vertex has six children, which are each themselves the root vertex of subtrees, or *subsidiaries*, of the rake tree.
4. The first vertex in each level of each subsidiary has two children.
5. Other vertices have one child.

In a rake tree, each vertex has a unique *rake-ID* which represents the topological position of the associated vertex in the rake tree, as shown in **Fig. 2**.

Definition A *rake-ID* is defined as a vector with three elements, denoted as (k, i, j):

1. k is the subsidiary number.
2. i is the level of the vertex in the subsidiary.
3. j represents the ordinal of the vertex among the vertices of the same level in its subsidiary. If a vertex is the first vertex of its level in the subsidiary, j is 1. If a vertex is the second vertex of its level in the subsidiary, j is 2.

The root vertex has the special rake-ID (0,0,1). The rake-ID of the fifth child of the root vertex, for example, is (5,1,1) and its children are (5,2,1) and (5,2,2), respectively. The rake tree can be equivalently redefined using rake-IDs.

Since a rake-ID specifies a particular vertex in the rake tree, it can also determine a particular vertex in HRTT. Therefore, by assigning a unique rake-ID to each sensor node, each sensor node is matched to a particular vertex in HRTT. The target position of each sensor node with rake-ID (k, i, j) in accordance with HRTT ((x_C, y_C), D, α, V) can be calculated as follows:

$$x = x_C + (i - j + 1)D\cos(\frac{\pi}{3}(k-1) + \alpha) + (j-1)D\cos(\frac{\pi}{3}k + \alpha)$$

$$y = y_C + (i - j + 1)D\sin(\frac{\pi}{3}(k-1) + \alpha) + (j-1)D\sin(\frac{\pi}{3}k + \alpha)$$

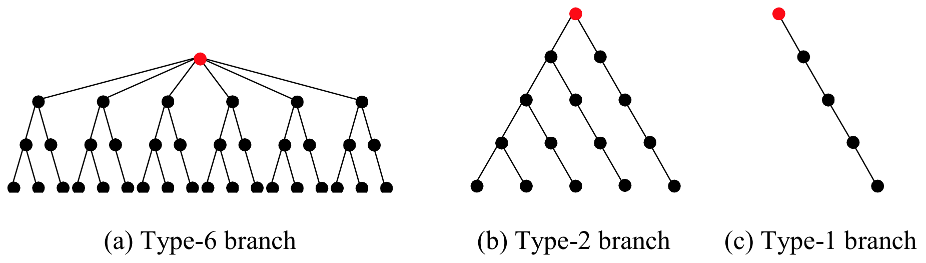


Fig. 3. Branch types of a rake tree

A rake tree can be divided into branches, which in turn can be divided into smaller branches. A branch of a rake tree can be represented with $((k, i, j), N)$ where (k, i, j) is the rake-ID of the head of the branch and N is the number of vertices in the branch. A branch is one of the three types, which are as follows:

- *Type-6 branch*: If $k = 0$ for a branch, then the head of the branch, which is incidentally the root vertex of the rake tree, has six children and the branch is shaped as shown in **Fig. 3a**.
- *Type-2 branch*: If $k \neq 0$ and $j = 1$ for a branch, then the head of the branch has two children and the branch is shaped as shown in **Fig. 3b**.
- *Type-1 branch*: Otherwise, the head of the branch has one child and the branch is shaped as shown in **Fig. 3c**.

The branches of a rake tree are units of division in a rake tree scheme. A rake tree scheme divides the target HRTT into several branches to match them to groups of sensor nodes.

A branch may contain several sub-branches. For each type of the branch, sub-branches of branch $((k, i, j), N)$ can be determined as follows:

- For a *Type-6 branch*, which can be given to only the root node, there are six sub-branches (**Fig. 3a**) which are $((1, 1, 1), (N-1)/6)$, $((2, 1, 1), (N-1)/6)$, …, $((6, 1, 1), (N-1)/6)$. In case $(N-1)/6$ is not an integer, the remnants are distributed starting from the first sub-branch so that only one level is appended to the bottom of the sub-branch. If more vertices are left, they are added to the vertices of the second sub-branch, the third sub-branch, and the next sub-branch until all the remaining vertices are distributed. By distributing the remaining vertices like this, the rake tree can be equivalent to the target HRTT, whereas the remaining vertices correspond to the farthest vertices of the target HRTT.
- For a *Type-2 branch*, there are two sub-branches (**Fig. 3b**) which are $((k, i+1, 1), N_1)$ and $((k, i+1, 2), N_2)$. N_1 and N_2 can be determined by solving the following simultaneous equations: $N-1 = N_1 + N_2$ and $N_1 = N_2(N_2+1)/2$. In case N_1 and N_2 are not integers, the remnants are added to N_1 in order to minimize the unbalance.
- For a *Type-1 branch*, there is only one sub-branch (**Fig. 3c**) which is $((k, i+1, j+1), N-1)$.

5.1.2 Matching with the Rake Tree Scheme

The divide-and-conquer strategy is employed to match the sensor nodes to vertices of the rake tree. As matching progresses from the root node toward the leaf nodes, the base tree is transformed into a rake tree. So, a subtree of the base tree is matched to a branch of the rake tree, as shown in **Fig. 4**.

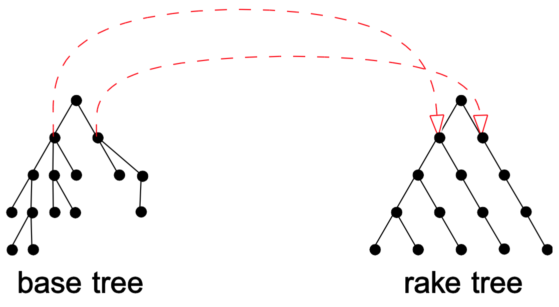


Fig. 4. Matching subtrees of a base tree to the branches of a rake tree

In matching with the rake tree scheme, the root node determines a rake tree from the target HRTT $((x_{root}, y_{root}), D, \alpha, V)$ at first. Then, the root node matches itself with a rake-ID and its children with branches. When matching at the root node is finished, matching at each child of the root node starts. After matching at the children of the root node is finished, matching at the grandchildren of the root node starts. It progresses towards the leaf nodes of the base tree.

Matching at node n is started when a branch $((k, i, j), N)$ is given to n by its parent, as follows:

1. Node n matches itself to the rake-ID of the head of the given branch, which is (k, i, j). If node n is the root node, it matches itself to the rake-ID (0, 0, 1).
2. The sub-branches of the given branch are determined by node n, as detailed in Section 5.1.1. In case of the root node, it divides the rake tree into six sub-branches except the root vertex, specifically $((1, 1, 1), (V - 1) / 6)$, $((2, 1, 1), (V - 1) / 6)$, …, $((6, 1, 1), (V - 1) / 6)$.
3. The number of children of node n in the base tree is adjusted to be the same as the number of the sub-branches in the rake tree. This is accomplished by *adjusting children*, as described later in this section. In case of the root node, it has six sub-branches and can only match its six children to the sub-branches. Therefore, it should adjust its own children in the base tree so that only six children are left to be matched.
4. Since the number of vertices of each sub-branch is already decided by the rake tree, each child of the node n should have the same number of descendants. This can be accomplished by *adjusting descendants*, which also will be discussed later in this section.
5. Each sub-branch is matched to each child of node n by sending a message that contains the representation of the sub-branch.

Once all the sensor nodes, up to the leaf nodes, are matched to their rake-IDs, the matching phase is complete.

5.1.2.1 Adjusting Children

Adjusting children is a process that adjusts the number of children of a sensor node in the base tree to match the number of sub-branches of a branch in a rake tree. The number of sub-branches, denoted as *TOC*, can be identified from the type of the branch. *TOC* = 6 for a

Type-6 branch, *TOC* = 2 for a *Type-2 branch*, and *TOC* = 1 for a *Type-1 branch*, respectively. *Adjusting children* algorithm is shown in Fig. 5. Unfortunately, for some rare cases, adjusting children may fail due to the lack of connectivity between the parent and the grandchildren. If this fails, a branch of the rake tree can be missing and the sensor nodes that were expected to be matched to the vertices of the branch will be matched to the vertices of other branches, incurring an unbalanced rake tree, and, eventually, an incomplete HRTT in which the boundary is not hexagonal.

5.1.2.2 Adjusting Descendants

Adjusting descendants is a process that adjusts the number of descendants of each child of a sensor node in the base tree to match the number of vertices in the sub-branch that will be given to the child. The number of descendants of child c, denoted as TOD_c, can be identified from the type and the number of vertices n of the branch as follows. For a *Type-6 branch*, $N - 1 = TOD_1 + TOD_2 + \dots + TOD_6$ and $TOD_1 = TOD_2 = \dots = TOD_6$. For a *Type-2 branch*, $N - 1 = TOD_1 + TOD_2$ and $TOD_1 = TOD_2 (TOD_2 + 1) / 2$. For a *Type-1 branch*, $N - 1 = TOD_1$.

For adjusting descendants, each child is checked with the number of descendants one by one. If the number of descendants of child c, NOD_c, in the base tree is smaller than TOD_c, one of the child c and the child $c + 1$ is adopted to the other with the same process as described above for adjusting children to effectively increase NOD_c by merging the subtree of c and the subtree of $c + 1$.

```
Procedure AdjustingChildren($n$, $NOC$, $TOC$, $childrenList$)
  $sortedChildrenList$ ← sort $childrenList$ w.r.t angles around node $n$
  $c_1$ ← get a node and delete it from $sortedChildrenList$
  if $NOC > TOC$ then
    $adjustedChildrenList$ ← Ø
    while $NOC > TOC$ and $sortedChildrenList \neq Ø$ do
      $c_2$ ← get a node and delete it from $sortedChildrenList$
      if $c_1$ is within communcation range from $c_2$ then
        if $c_1$ is at a greater distance from node $n$ than $c_2$ then
          add $c_1$ as a new child of $c_2$
          $adjustedChildrenList$ ← $adjustedChildrenList \cup \{c_2\}$
        else
          add $c_2$ as a new child of $c_1$
          $adjustedChildrenList$ ← $adjustedChildrenList \cup \{c_1\}$
        end if
        $NOC$ ← $NOC - 1$
        $c_1$ ← get a node and delete it from $sortedChildrenList$
      else
        $c_1$ ← $c_2$
      end if
    end while
  else if $NOC < TOC$ then
    $adjustedChildrenList$ ← $childrenList$
    while $NOC < TOC$ and $c_1 \neq Ø$ do
      $grandchildrenList$ ← get the children list of $c_1$
      while $NOC < TOC$ and $grandchildrenList \neq Ø$ do
        $c_g$ ← get a node and delete it from $grandhildrenList$
        if $c_g$ is within communcation range from node $n$ and $c_g$ is not in
        $adjustedChildrenList$ then
          get rid of $c_g$ from child list of $c_1$
          $adjustedChildrenList$ ← $adjustedChildrenList \cup \{c_g\}$
          $NOC$ ← $NOC + 1$
        end if
      end while
      $c_1$ ← get a node and delete it from $sortedChildrenList$
    end while
  end if
  return $adjustedChildrenList$
end procedure
```

Fig. 5. *AdjustingChildren* algorithm

On the other hand, if NOD_c is larger than TOD_c, the subtree of c is split so that only TOD_c is left in the subtree. This can be done using the *CutSubtree* algorithm (**Fig. 6**) by calling it with *CutSubtree*(c, $NOD_c - TOD_c$). It then finds the sensor nodes among the descendants of c that can be adopted by other nodes in the descendants of $c + 1$. The found sensor nodes are adopted with the same process as described for adjusting children in order to effectively reduce NOD_c. An example of adjusting descendants is illustrated in **Fig. 7**.

```
Procedure CutSubtree($n$, $numToCut$)
  $numSelected \leftarrow 0$
  $adopteeList \leftarrow \emptyset$
  $adoptorList \leftarrow \emptyset$
    while $numSelected < numToCut$ do
    $kid \leftarrow n$
    $FOUND \leftarrow$ false
      repeat
        $kid \leftarrow$ the last child of $kid$
        if $kid$ has a neighobor in the next subtree and $NOD_{kid} \leq numToCut - numSelected$
          then
          $FOUND \leftarrow$ true
          $adoptorList \leftarrow adoptorList \cup \{kid\}$
          $adoptorList \leftarrow adoptorList \cup$ {the neighbor in the next subtree}
          $numSelected \leftarrow numSelected + NOD_{kid}$
        endif
      until $FOUND$
    end while
  return $adopteeList$, $adoptorList$
end procedure
```

Fig. 6. *CutSubtree* algorithm

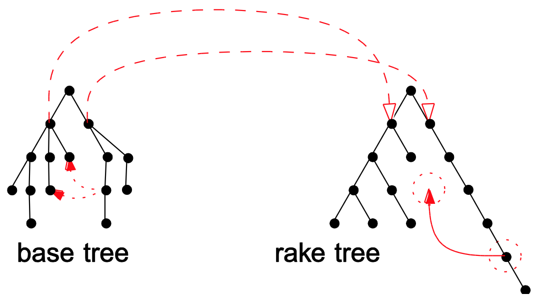


Fig. 7. An example of adjusting descendants

With a sparse initial layout, there may be such cases that *CutSubtree* cannot find any descendant to be adjusted. In these cases, the vertices in the sub-branches will not be the same as *TOD* and the branch may be unbalanced. It eventually may make the rake tree unbalanced and the final layout may not have the boundary of the target HRTT as determined by the root node.

5.2 Balanced Rake Tree Scheme

A balanced rake tree scheme is proposed in order to build a balanced rake tree even when the rake tree scheme fails. It rematches the sensor nodes that are originally matched to a vertex that is outside the balanced rake tree to another vertex inside the balanced rake tree.

5.2.1 Balanced Rake Tree

Definition A *balanced rake tree* is a type of rake tree with the following properties:

1. The differences between the values of *i* in the rake-IDs of all the leaf vertices are less than or equal to 1.

2. The maximum values of i in the rake-IDs in subsidiary, s_1, is not greater than the maximum value of i in subsidiary s_2, if $s_1 > s_2$.
3. In the maximum level of a subsidiary, the values of j are consecutive starting from 1.

The second property implies that the maximum level of a subsidiary is not greater than the maximum level of the preceding subsidiaries. Also, the third property implies that the leaf vertices at the maximum level are gathered at the left side whereas other leaf vertices are gathered at the right side. On the whole, unlike general rake tree, a balanced rake tree always has the same boundary as the HRTT.

In order to build a balanced rake tree, the following process is added at the last movement phase of the rake tree scheme:

1. At the start of the movement phase, each node u can verify whether its rake-ID (k_u, i_u, j_u) is included in the balanced rake tree where u denotes the node. Specifically speaking, if i_u is greater than i_{max}, which is the maximum value of i in the subsidiary, then it is not included in the balanced rake tree and the node is referred as an *unfit node*. i_{max} can be derived from the total number of sensor nodes in the subsidiary.
2. If a node recognizes itself as an unfit node, it determines its alternative vertex in the balanced rake tree rather than the original one. The alternative target position is the position of the last rake-ID (k_u, i_l, j_l) in the balanced rake tree, which is most probably not assigned to any node. i_l is the same as i_{max}. j_l is the biggest value of j among the rake-IDs with i_l in the subsidiary. j_l also can be derived from the number of sensor nodes in the subsidiary.
3. Once the alternative vertex is determined, the unfit node moves to the position corresponding to the alternative vertex and checks if the position is already occupied by another sensor node by exchanging messages.
4. If the position is already occupied, it then moves to the position corresponding to the vertex of the next rake-ID. The next rake-ID is (k_u, i_l, $j_l - 1$) or (k_u, $i_l - 1$, $i_l - 1$) if $j_l = 1$. If the next position is also occupied, it continues to move to the next position until an unoccupied position is encountered.

The balanced rake tree creates the vertices to be as many as the sensor nodes allocated in the subsidiary. As the unfit nodes are not assigned with rake-IDs that are within the balanced rake tree, the same number of unassigned rake-IDs must exist in the balanced rake tree as the unfit nodes. Thus, all the unfit nodes eventually succeed in finding unoccupied positions.

Even though the balanced rake tree scheme presents a simple and effective solution to build a balanced rake tree, it may fail to do so if the root node fails to group its descendants with the exact number of sensor nodes. In such a case, the unfit nodes cannot move to another subsidiary with this scheme. The final layout can have the subsidiaries with unbalanced sizes while each of the subsidiaries is perfectly balanced on its own.

5.3 Rake-ID Space Scheme

In order to solve the matching problem, the rake-ID space scheme takes a different approach from the rake tree and balanced rake tree schemes. Instead of transforming the base tree into a rake tree, rake-ID space scheme divides the balanced rake tree so that the divided parts can be exactly matched to subtrees of the base tree. A *rake-ID space* is used to represent a group of vertices that are adjacent in a balanced rake tree. It deals with vertices rather than rake-IDs to facilitate geometrical operations such as division into subspaces of adjacent vertices and extraction of a single vertex from a rake-ID space to be matched to a sensor node.

Rake-ID space exploits the geometrical relationships between vertices in a balanced rake tree to represent them with a limited number of variables. Therefore, by adopting rake-ID space, a group of vertices can be represented with only a constant amount of memory and can be transmitted with only a short message.

5.3.1 Rake-ID Space

A rake-ID space describes the adjacent vertices of the balanced rake tree that can reside on more than one of consecutive subsidiaries. An example of a rake-ID space of all vertices in a balanced rake tree except the root vertex is shown in **Fig. 8**. The subsidiaries are separated from the original balanced rake tree and put next to each other in the rake-ID space. Each subsidiary is shown as dots arranged in a triangle where each dot represents a vertex in the balanced rake tree. A rake-ID space can have an uneven bottom, as shown in **Fig. 9**. The dots at the lowest level of a rake-ID space represent the leaf vertices of the balanced rake tree. Following the definition of a balanced rake tree, the dots at the lowest level are poised at the left-most side of the bottom.

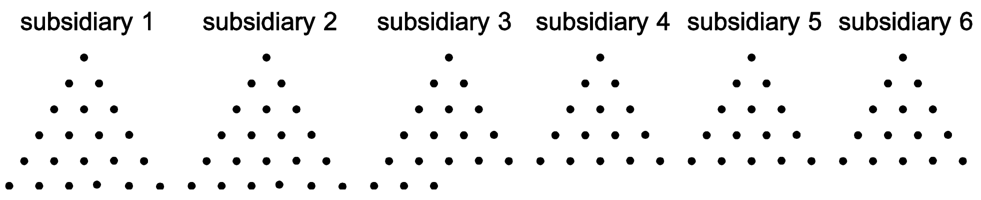


Fig. 8. An example of a rake-ID space of 105 vertices

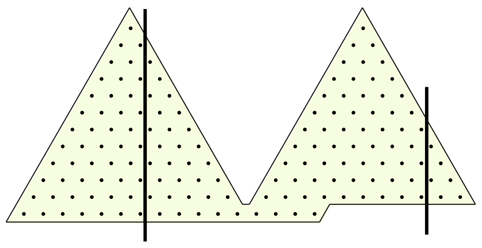
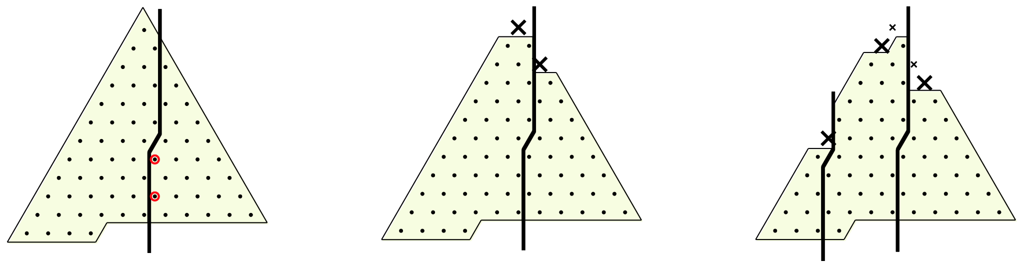


Fig. 9. A rake-ID space of 148 vertices divided into subspaces of 48, 94, and 6 vertices, respectively

A rake-ID space can be divided into several subspaces with vertical walls, as shown in **Fig. 9**. Each subspace is transmitted to a sensor node for matching the included vertices to its descendants. The division of a rake-ID space should produce the given number of subspaces according to the number of children of the current sensor node. Each of the subspaces should have the given number of vertices according to the number of descendants of each child. So, in order to divide a rake-ID space into subspaces with an arbitrary number of vertices, vertical walls can be bent in the middle. In the case of **Fig. 10a**, the rake-ID space of 70 vertices is divided into two subspaces of 43 and 27 vertices respectively by bending the vertical wall in the middle. After a rake-ID space is given to a sensor node, the sensor node can take a vertex from the top of the rake-ID space to match itself, as shown in **Fig. 10b**. If there is more than one vertex in the top level, the left-most one is taken, as shown by the second subspace of **Fig. 10c**.



(a) A rake-ID space divided into subspaces of 43 and 27 vertices respectively

(b) A vertex taken from each subspace

(c) A vertex taken from each subspace after additional division

Fig. 10. A rake-ID space of 148 vertices divided into subspaces of 48, 94, and 6 vertices, respectively

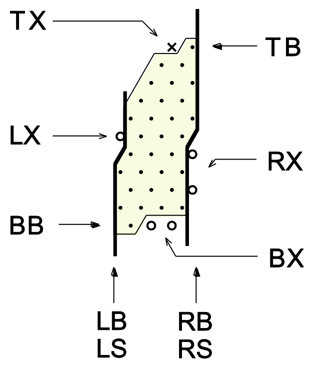


Fig. 11. Elements of rake-ID space

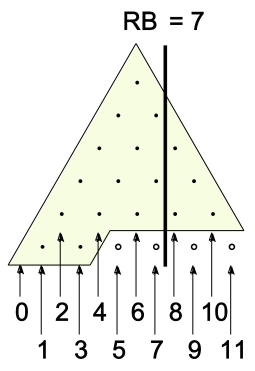


Fig. 12. Position of vertical walls

A rake-ID space can be denoted with a 10-element vector *RIS* = (*LS*, *RS*, *LB*, *LX*, *RB*, *RX*, *TB*, *TX*, *BB*, *BX*) as depicted in **Fig. 11**. The meaning of each element is described as follows. *LS* and *RS* are the left-most subsidiary and right-most subsidiary that the vertices in the rake-ID space resides on. *LB* and *RB* are positions of the left wall in the left-most subsidiary and the right wall in the right-most subsidiary. *LX* and *RX* are the numbers of excluded vertices among the left-most and right-most vertices, which are included in other rake-ID spaces. *TB* and *BB* are the top level and bottom level respectively. *TX* is the number of excluded vertices among top-level vertices, which are the rake-IDs already taken. Lastly, *BX* is the number of excluded vertices among the bottom-level vertices. With *LB* and *RB*, the position of a vertical wall is specified as shown in **Fig. 12** with respect to the bottom level even if there are no vertices in the bottom level.

A rake-ID space can be created, given the number of vertices. At the time of creation, a rake-ID space contains six subsidiaries. For example, a rake-ID space created with 105 vertices, as shown in **Fig. 8**, has *LS* = 1, *RS* = 6, *LB* = 0, *LX* = 0, *RB* = 11, *RX* = 0, *TB* = 1, *TX* =

0, *BB* = 6, and *BX* = 18. *LS* = 1 and *RS* = 6 represent that six subsidiaries are contained in the rake-ID space. *RB* = 11 represents the position of the right-most vertices in the sixth subsidiary with respect to the bottom level, which, in this case, has no vertex according to the value of *BX*.

5.3.1 Matching with the Rake-ID Space Scheme

The rake-ID space scheme also employs the divide-and-conquer strategy in the matching phase. At each sensor node, given with a rake-ID space, the rake-ID space is divided into subspaces to be given to the children of the sensor node.

At first, the root node matches itself to rake-ID (0, 0, 1) and creates a rake-ID space, *RIS*, with the total number of its descendant sensor nodes. *RIS* contains the vertices of a rake tree that are equivalent to the target HRTT except for the root vertex. Then, the root node sends *HRTT* and RIS_c to its *c*th child. Each sensor node *n* that received *HRTT* and RIS_n takes a rake-ID from RIS_n and modifies RIS_n into RIS_n'. Then, it divides RIS_n' and sends the subspaces to its children. Eventually, all sensor nodes get their rake-IDs from the rake-ID spaces, which match them to vertices in the target layout.

In the rake-ID space scheme, node *n* matches its children as follows:

1. Node *n* except the root node takes a rake-ID (*k*, *i*, *j*) from *RIS*, which turns it into *RIS'* and matches itself to the rake-ID (*k*, *i*, *j*). The root node takes a rake-ID (0, 0, 1).
2. Node *n* divides *RIS'* into subspaces RIS_1, RIS_2, …, RIS_g which have N_{c1}, N_{c2}, …, N_{cg} vertices respectively where *g* is the number of children of node *n*; c_1, c_2, …, c_g are the children of node *n*, sorted with respect to the angles around node *n*; and N_{c1}, N_{c2}, …, N_{cg} are the numbers of descendants of c_1, c_2, …, c_g.
3. Node *n* sends RIS_i to each child c_i.

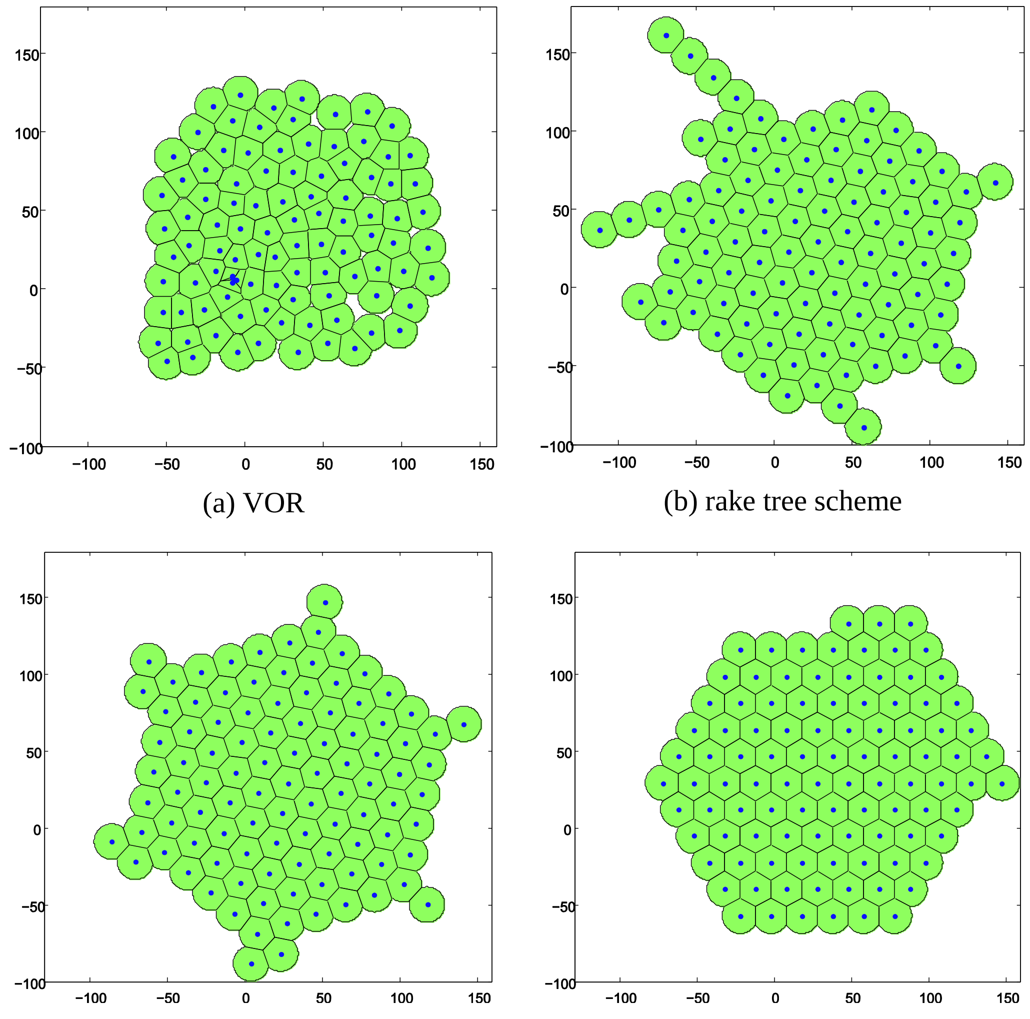
Once all sensor nodes, including the leaf nodes, are matched to their rake-IDs, the matching phase is complete.

6. Performance Evaluation

The performance of the tree-based approach adopting an HRTT layout model with three matching schemes is evaluated using the Matlab simulator [25]. Total area of coverage, average moving distance per sensor node, and gain of coverage per unit moving distance are measured. For comparison, VEC, VOR, and Minimax are also evaluated with the same environment [3]. The number of iterations of VEC, VOR, and Minimax is limited to 100, over which the coverage area is not notably improved. We set the sensing range (R_S) to 11.547*m* and the minimum communication range (R_C) to 20*m* ($= \sqrt{3} \cdot R_S$).

Examples of the final layout of several deployment schemes are shown in **Fig. 13**. These had 100 sensor nodes and the same initial layout within 2,500m^2. In the example results, VOR shows an irregular pattern even bearing small holes inside and the movement of the sensor nodes is still ongoing even after 100 iterations. In contrast, the tree-based approach with the rake tree scheme shows a partially optimal layout with an irregular boundary. Moreover, the tree-based approach the adopting balanced rake tree and rake-ID space schemes accomplished the optimal layout.

Fig. 14 and **Fig. 15** compare the total area of coverage of the final layouts. In **Fig. 14**, the area of initial layout is fixed as 2,500m^2 and 10,000m^2 whereas the number of sensor nodes varies. On the other hand, **Fig. 15** shows the total area of coverage with 100 sensor nodes whereas the area of initial layout varies. The coverage area of the tree-based approach adopting the rake-ID space scheme (denoted as RSpace in the graphs) is actually equal to the optimal coverage that is accomplished with HRTT. This attests that successfully deploys the sensor nodes into the target layout. By the way, when the area of initial layout is 10,000m^2 and the number of sensor nodes is below 100, the balanced rake tree scheme (denoted as BRake in the graph) shows lower coverage than the rake-ID space scheme. This is because the root node has failed to exactly adjust the number of descendants for each child to the number of vertices in the subsidiary to be matched due to the lack of connectivity.

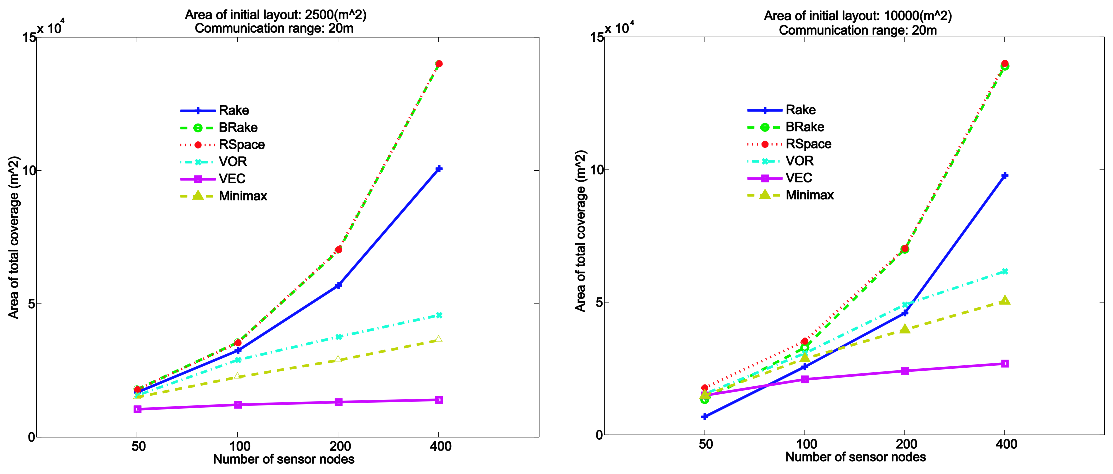


(a) VOR (b) rake tree scheme

(c) balanced rake tree scheme (d) rake-ID space scheme

Fig. 13. Examples of deployment result

The tree-based approach adopting the rake tree scheme (denoted as Rake in the graph) results in lower coverage than the balanced rake tree and rake-ID space schemes. For fairness, we did not take into account the area covered by sensor nodes that moved out of the target HRTT as well as their moving distance. Therefore, the final layouts of the rake tree scheme that constructs unbalanced rake trees have smaller coverage areas than the other matching schemes of the tree-based approach even though there are exactly the same number of sensor nodes actually covering the same area. In spite of that, all the tree-based approaches cover more area than the final layouts of VEC, VOR, and Minimax when the area of initial layout is below 2,500m^2. The coverage area of the rake tree scheme gets lower when the area of initial layout is larger or the number of sensor nodes is smaller. This is because the unbalance of the rake trees becomes more severe when the density and the degree of connectivity become lower in the initial layout.



(a) initial layout of 2,500m^2 area (b) initial layout of 10,000m^2 area

Fig. 14. Coverage area of the final layout depending on number of sensor nodes

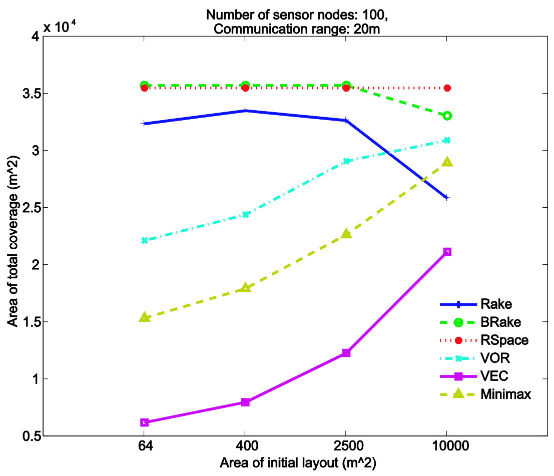


Fig. 15. Coverage area of the final layout depending on initial layout areas

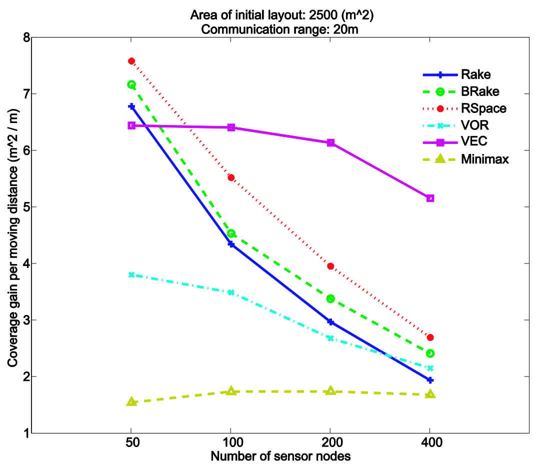


Fig. 16. Coverage gain per movement

In contrast to the tree-based approach, VEC, VOR, and Minimax fail to disperse the sensor nodes until they cover the maximum area. Their coverage areas do not sufficiently increase as much as the number of sensor nodes or iterations increases. In addition, they get relatively less coverage area when the number of sensor nodes is larger or the initial layout is smaller.

Accordingly, we can find that VEC, VOR, and Minimax are not efficient for concentrated or dense initial layout.

To estimate how efficiently the sensor nodes move with each scheme, the improvement of coverage area per unit distance of movement is shown in **Fig. 16**. Overall, the rake tree, balanced rake tree, and rake-ID space schemes work more efficiently than VOR and Minimax. This is because, in VOR and Minimax, the sensor nodes inside fluctuate until the sensor nodes outside move further outward and make room for other sensor nodes to move to. VEC works more efficiently than any other schemes only by moving a small distance rather than enlarging the coverage area.

7. Conclusion

In the sensor networks, the deployment has a critical effect on the performance and lifetime. However, existing deployment methods do not successfully address the important issues of building the optimal final layout or minimizing the energy consumption for the sensor movement. In this paper, a novel approach, namely a *tree-based approach*, is proposed to better address such issues. In the tree-based approach, we separated the modeling of the target layout from the deployment problem. Therefore, diverse sets of requirements for sensor networks can be embraced without changing the deployment scheme. Moreover, we defined a *hexagonal-boundary regular triangular tessellation* (HRTT) layout model. It covers the maximum area with a given number of sensor nodes while preserving robust connectivity when $R_C \geq \sqrt{3} \cdot R_S$.

At first, tree-based approach for deployment problem builds a spanning tree, called base tree. Through the base tree, the global informations about the initial layout of the sensor network are collected and used for realization of the specified layout model, which determines the target layout. Then, tree-based approach attempts to match each sensor node to a vertex in the target layout with divide-and-conquer strategy. The sensor nodes can determine their target positions with the matched vertices and find the shortest routes to the target positions which consume minimum energy. Once all the sensor nodes move along the shortest routes and arrive at the target positions, the target layout is exactly achieved as determined from the layout model.

For the tree-based approach, we also proposed three simple and effective matching schemes: *rake tree scheme*, *balanced rake tree scheme*, and *rake-ID space scheme*. They match sensor nodes to the vertices in the layout. The proposed matching schemes guarantee to achieve the target layout, whereas previous works do not. In addition, these schemes result in smaller energy consumption than previous works.

In this paper, we suggested a new perspective on the sensor network deployment problem, which can be viewed as a minimum-weight complete matching problem. With this perspective, further study on that problem may reveal a better matching scheme. In addition, the update of sensor nodes after first deployment is also an interesting issue. Therefore, we will deeply handle the update method of sensor nodes on the further study. Lastly, we will also conduct the systematical analysis of our proposed schemes for the future work.

References

[1] V. C. Gungor and G. P. Hancke, “Industrial wireless sensor networks: challenges, design principles, and technical approaches,” *IEEE Trans. on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265,

2009. Article (CrossRef Link)
[2] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *Proc. of INFOCOM 2003*, vol. 2, pp. 1293–1303, 2003. Article (ieeexplore Link)
[3] G.Wang, G. Cao and T. L. Porta, "Movement-assisted sensor deployment," *IEEE Trans. on Mobile Computing*, vol. 5, no. 6, pp. 640–652, 2006. Article (CrossRef Link)
[4] N. Heo and P. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 35, no. 1, pp. 78-92, 2005. Article (CrossRef Link)
[5] M.-l. Lam and Y.-h. Liu, "ISOGRID: an efficient algorithm for coverage enhancement in mobile sensor networks," in *Proc. of IROS 2006*, pp. 1458–1463, 2006. Article (ieeexplore Link)
[6] G. Song, W. Zhuang and A. Song, "Self-deployment of mobile sensor networks in complex indoor environments," in *Proc. of WCICA 2006*, vol. 1, pp. 4543–4546, 2006. Article (ieeexplore Link)
[7] M. Ma and Y. Yang, "Adaptive triangular deployment algorithm for unattended mobile sensor networks," *IEEE Trans. on Computers*, vol. 56, no. 7, pp. 946–847, 2007. Article (CrossRef Link)
[8] G. Tan, S. Jarvis and A.-M. Kermarrec, "Connectivity-guaranteed and obstacle-adaptive deployment schemes for mobile sensor networks," *IEEE Trans. on Mobile Computing*, vol. 8, no. 6, pp. 836–848, 2009. Article (CrossRef Link)
[9] S. Yang, M. Li and J.Wu, "Scan-based movement-assisted sensor deployment methods in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1108–1121, 2007. Article (CrossRef Link)
[10] P.-C. Wang, T.-W. Hou and R.-H. Yan, "Maintaining coverage by progressive crystal-lattice permutation in mobile wireless sensor networks," in *Proc. of ICSNC 2006*, pp. 42–42, 2006. Article (ieeexplore Link)
[11] X. Yu, W. Huang, J. Lan and X. Qian, "A novel virtual force approach for node deployment in wireless sensor network," in *Proc. of DCOSS*, pp. 359–363, 2012. Article (ieeexplore Link)
[12] M.-l. Lam and Y.-h. Liu, "Heterogeneous sensor network deployment using circle packings," in *Proc. of ICRA 2007*, pp. 4442–4447, 2007. Article (ieeexplore Link)
[13] G. M. Hoffmann and C. J. Tomlin, "Mobile sensor network control using mutual information methods and particle filters," *IEEE Trans. on Automatic Control*, vol. 55, no. 1, pp. 32–47, 2010. Article (CrossRef Link)
[14] M. Zhao and Y. Yang, "Optimization-Based Distributed Algorithms for Mobile Data Gathering in Wireless Sensor Networks," *IEEE Trans. on Mobile Computing*, vol. 11, no. 10, pp. 1464–1477, 2012. Article (CrossRef Link)
[15] R. V. Kulkarni and G. K. Venayagamoorthy, "Particle swarm optimization in wireless-sensor networks: A brief survey," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 41, no. 2, pp. 262–267, 2011. Article (CrossRef Link)
[16] G. A. Montoya, C. Velasquez-Villada and Y. Donoso, "Energy optimization in mobile wireless sensor networks with mobile targets achieving efficient coverage for critical applications," *International Journal of Computers Communications & Control*, vol. 8, no. 2, pp. 247–254, 2013. Article (ijccc Link)
[17] M. Bui, F. Butelle and C. Lavault, "A distributed algorithm for constructing a minimum diameter spanning tree," *Journal of Parallel and Distributed Computing*, vol. 64, no. 5, pp. 571 – 577, 2004. Article (CrossRef Link)
[18] F. Dressler, "A study of self-organization mechanisms in ad hoc and sensor networks," *Computer Communications*, vol. 31, no. 13, pp. 3018 – 3029, 2008. Article (CrossRef Link)
[19] A. Nayak and I. Stojmenovic, "Wireless Sensor and Actuator Networks: Algorithms and Protocols for Scalable Coordination and Data Communication," *Wiley*, 2010. Article (CrossRef Link)
[20] X. Bai, D. Xuan, Z. Yun, T. H. Lai and W. Jia, "Complete optimal deployment patterns for full-coverage and k-connectivity ($k \leq 6$) wireless sensor networks," in *Proc. of MobiHoc 2008*, pp. 401–410, 2008. Article (ACM Link)
[21] B. Wang, "Coverage Control in Sensor Networks," *Computer Communications and Networks*, Springer, 2010. Article (CrossRef Link)
[22] X. Bai, Z. Yun, D. Xuan, T. Lai and W. Jia, "Optimal patterns for four connectivity and full

coverage in wireless sensor networks," *IEEE Trans. on Mobile Computing*, vol. 9, no. 3, pp. 435-448, 2010. Article (CrossRef Link)

[23] P. Vaidya, "Geometry helps in matching," in *Proc. of STOC 1988*, pp. 422–425, 1988. Article (SIAM Link)

[24] H. Ghim, N. Kim, D. Kim, M. Choi and H. Yoon, "An energy-efficient dispersion method for deployment of mobile sensor networks," *IEICE Electronics Express*, vol. 7, no. 10, pp. 722–727, 2010. Article (CrossRef Link)

[25] http://www.mathworks.com/products/matlab/

Hojin Ghim received the B.S. degree in Computer Engineering from Hongik University, Korea, in 2002, and the M.S. degree and the Ph.D. degree in Computer Science from KAIST in 2004 and 2010, respectively. Since 2010, he has been a software engineer of LG Electronics. His research interests include mobile networks and mobile user interaction.

Dongwook Kim received his B.S. degree in the Information and Computer Engineering from AJOU University, South Korea, in 2002 and his M.S. and Ph. D. degrees in the Computer Science from KAIST, South Korea, in 2004 and 2009, respectively. Since 2009, He is a senior engineer at the Network Business Division, Samsung Electronics. His research interests include the design and optimization of LTE (Long Term Evolution)-advanced systems with SON (Self-Organizing Networks) technologies, optimal deployment of mobile sensor networks, broadcast techniques for sensor networks, and optimal handover schemes for 4G mobile systems.

Namgi Kim received the B.S. degree in Computer Science from Sogang University, Korea, in 1997, and the M.S. degree and the Ph.D. degree in Computer Science from KAIST in 2000 and 2005, respectively. From 2005 to 2007, he was a research member of the Samsung Electronics. Since 2007, he has been a faculty of the Kyonggi University. His research interests include sensor system, wireless system, and mobile communication.