Optimal Movement-Assisted Sensor Deployment and Its Extensions in Wireless Sensor Networks *

Jie Wu and Shuhui Yang
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Abstract

In wireless sensor networks (WSNs), a good sensor deployment method is vital to the quality of service (QoS) provided by WSNs. This QoS depends on the coverage of the monitoring area. In WSNs with locomotion facilities, sensors can move around and self-deploy to ensure coverage and load balancing. In SMART [1], various optimization problems are defined to minimize different parameters. In this paper, we focus on minimizing total moving distance and propose an optimal, but centralized solution, based on the Hungarian method. We then propose several efficient, albeit non-optimal, distributed solutions based on the scan-based solution in [1]. Extensive simulations have been done to verify the effectiveness of the proposed solutions.

Keywords: Hungarian method, load balance, sensor coverage, sensor deployment, wireless sensor networks (WSNs).

1 Introduction

The efficiency of a WSN depends on the coverage of the monitoring area. In many environments, such as remote harsh fields or disaster areas, sensor deployment cannot be performed manually or precisely. In addition, sensors may run out of battery after a certain time, requiring others to be moved to cover the coverage holes created by the dead sensors. In these cases, it is necessary to make use of mobile sensors [2] to provide the required coverage and load balancing.

In general, two methods can be used to enhance the coverage: *incremental sensor deployment* and *movement-assisted sensor deployment*. Incremental self-deployment [3] incrementally deploys additional sensors, usually one-at-a-time, with each node using data gathered from previously deployed nodes to determine its optimal location. Movement-assisted sensor deployment [4], on the other hand, uses a potential-field-based approach to move existing sensors by treating sensors as virtual particles, subject to virtual forces [5, 6, 7].

In this paper, we focus on load balancing solutions in WSN that minimize total moving distance of sensors. By load balancing, we mean each unit of monitoring area is covered by the same number of sensors. The monitoring area is a 2-D grid-based mesh (2-D mesh). We first provide an optimal solution in 2-D meshes. This solution is based on the classic Hungarian method, but requires global information. We then enhance the scan-based solution without resorting to global information, but with relatively competitive results in terms of the total moving distance.

The contributions of this paper are: (1) We systematically discuss the drawback of existing movement-assisted sensor deployment in WSNs. (2) We propose an optimal load balancing solution based on the Hungarian method that achieves minimum total moving distance and the number of moves. (3) We extend the scan-based solution to reduce total moving distance without resorting to global information. (4) We present several further extensions and discuss various trade-offs among total moving distance, number of moves, and converging speed. (5) We conduct extensive simulations and compare results of the proposed extended scan-based solutions with the optimal solution.

The following assumptions are used in this paper: (1) The monitoring and deployment area is an $n \times n$ grid, with each grid of size $r \times r$. In a 2-D mesh, each grid point at position (i, j) has four neighbors at positions: $(i-1, j)$, $(i, j-1)$, $(i, j+1)$, and $(i+1, j)$. Among existing approaches, TTDD [8] and GAF [9] use geographic location to partition the network into a 2-D mesh. (2) Each sensor has position information and has uniform sensing range $\sqrt{2}r$ and two transmission ranges $\sqrt{2}r$ (for intra-grid communication) and $\sqrt{5}r$ (for inter-grid communication). (3) The sensor network is sufficiently dense so that each grid

*This work was supported in part by NSF grants ANI 0073736, EIA 0130806, CCR 0329741, CNS 0422762, CNS 0434533, and CNS 0531410. Email: jie@cse.fau.edu, syang1@fau.edu.

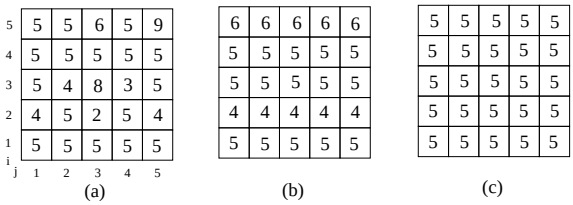


Figure 1. SMART: (a) initial deployment, (b) row scan, and (c) column scan.

point (cluster) has at least one sensor. Each grid point has one leader (clusterhead) to coordinate activities with leaders of four neighbors.

2 Preliminaries and Related Work

2.1 Movement-assisted deployment

The sensor placement issue has been widely studied recently [10]. Two methods can be used to enhance the coverage: incremental sensor deployment and movement-assisted sensor deployment.

In incremental sensor deployment in [3], nodes are deployed one by one, using the location information of previously deployed nodes to deploy the current one. This algorithm is not scalable and is computationally expensive.

[7] proposed a centralized virtual force based mobile sensor deployment algorithm (VFA), which combines the idea of potential field and disk packing [11]. In VFA, there is a powerful clusterhead, which will communicate with all the other sensors, collect sensor position information, and calculate forces and desired position for each sensor.

[5] developed a novel distributed self-deployment protocol for mobile sensors. They used Voronoi diagrams [12] to find coverage holes in the sensor network, and proposed algorithms to guide sensor movement toward the coverage hole. When applied to randomly deployed sensors, these algorithms can provide high coverage within a short time and limited moving distance. If the initial distribution of the sensors is extremely uneven, disconnection may occur, thus, the Voronoi polygon constructed may not be accurate enough, which results in more moves and larger moving distance. They adopted the optimization of random scattering of some sensors to cover holes. The termination condition of their algorithms is coverage instead of load balancing. [6] further explored the motion capability of sensors for relocation to deal with sensor failure or respond to new events.

Table 1. The scan process.

i	1	2	3	4	5
w_i	5	4	8	3	5
v_i	5	9	17	20	25
$\overline{v_i}$	5	10	15	20	25

2.2 SMART: a scan-based approach

SMART [1] is a hybrid of local and global approach. The sensor network is partitioned into an $n \times n$ 2-D mesh of grids. Each leader, in charge of communication with adjacent grids, knows the following information: (1) its grid's position, i, in the currently processed row/column of the 2-D mesh, and (2) the number of sensors, w_i, in the grid.

In SMART, the 2-D mesh is partitioned into 1-D arrays by row and by column. Two scans are used in sequence: one for all rows, followed by the other for all columns. Consider the 1-D array of grids where grid ID is labeled following the sequence in the linear line. Let v_i be the prefix sum of the first i grids, i.e., $v_i = \sum_{j=1}^{i} w_j$. Then $v_n = \sum_{j=1}^{n} w_j$ is the total sum. Clearly, $\overline{w} = v_n/n$ is the average number of sensors in a balanced state, and $\overline{v_i} = i\overline{w}$ is the prefix sum in the balanced state. Note that $\overline{w}$ is a real number which should be rounded to an integer $\lfloor \overline{w} \rfloor$ or $\lceil \overline{w} \rceil$. In a balanced state, $|w_i - w_j| \leq 1$ for any two grids in the 1-D array.

The scan algorithm works from one end of the array to another (first scan) and then from the other end back to the initial end (second scan). The first sweep calculates the prefix sum v_i, where each clusterhead i determines its prefix sum v_i by adding $v_{i-1} + w_i$ and forwarding v_i to the next grid. The clusterhead in the last grid determines v_n and $\overline{w} = v_n/n$ (load in a balanced state) and initiates the second scan by sending out $\overline{w}$. During this scan, each clusterhead determines $\overline{v_i} = i\overline{w}$ (load of prefix sum in a balanced state) based on $\overline{w}$ passed around and its own grid position i.

Knowing the load in the balanced state, each grid can easily determine its "give/take" state. Specifically, when $w_i - \overline{w} = 0$, grid i is in the "neutral" state. When $w_i - \overline{w} > 0$, it is overloaded and in the "give" state; and when $w_i - \overline{w} < 0$, it is underloaded and in the "take" state. Each grid in the give state also needs to determine the number of sensors (load) to be sent to each direction: $w_i^{\rightarrow}$ for load in the positive direction (or simply give-right) and ${}^{\leftarrow}w_i$ for load in the negative direction (give-left).

Based on the scan procedure, it is clear that

$$w_i^{\rightarrow} = \min\{w_i - \overline{w}, \max\{v_i - \overline{v_i}, 0\}\} \quad (1)$$

$${}^{\leftarrow}w_i = (w_i - \overline{w}) - w_i^{\rightarrow} \quad (2)$$

The 2-D scan process involves a row scan followed by a column scan as shown in Figures 1 (b) and (c), respectively. Table 1 shows details of the row scan on the third row where

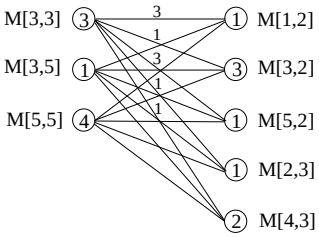


Figure 2. The node and edge weighted bipartite graph of Figure 1, "give" grids on left and "take" grids on right.

i is the column number in Figure 1. Only the grid at column 3 is in the "give" state, since its load is higher than $\overline{w} = 5$. For column 3, $w_3^{\rightarrow} = 2$ will be assigned to column 2 and ${}^{\leftarrow}w_4 = 1$ will be assigned to column 2. Similarly, a set of conditions can be given for a "take" state: $w_i^{\leftarrow}$ for take-right and ${}^{\rightarrow}w_i$ for take-left.

$$ {}^{\rightarrow}w_i = \min\{\overline{w} - w_i, \max\{v_{i-1} - \overline{v_{i-1}}, 0\}\} \quad (3) $$

$$ w_i^{\leftarrow} = (\overline{w} - w_i) - {}^{\rightarrow}w_i \quad (4) $$

The result of the 2-D scan process usually does not generate an ideal global balanced state as in Figure 1. However, the maximum load difference between any two grids is bounded by 2. It is shown that the scan-based approach is optimal (in terms of both total moving distance and number of moves) for 1-D arrays, but not for 2-D meshes.

Example 1: Consider a 2×2 mesh $M[1,1] = 3, M[1,2] = 1, M[2,1] = 3$, and $M[2,2] = 5$. A scan on rows will change load distribution of the mesh to $M[1,1] = 2, M[1,2] = 2, M[2,1] = 4$, and $M[2,2] = 4$, and a scan on columns will balance the mesh to $M[1,1] = 3, M[1,2] = 3, M[2,1] = 3$, and $M[2,2] = 3$. A total of 4 moves occur, however, the optimal solution requires only 2 moves from $M[2,2]$ to $M[1,2]$ directly.

Example 2: Consider a large 2-D mesh where all nodes have a load of 2 except $M[i,j] = 3$, $M[i,j+d] = 1$, $M[i+1,j] = 1$, and $M[i+1,j+d] = 3$. A row scan will balance the mesh with a total moving distance $2d$, while a column scan will balance the mesh in an optimal way using a total moving distance 2.

3 An optimal solution

3.1 Hungarian method

Let us consider the *edge weighted matching problem* in a complete bipartite graph $K_{m,m}$ with each edge associated with a number called its weights. The objective is to find a perfect matching (of m pairs), such that the sum of the weights of edges in the matching is minimum.

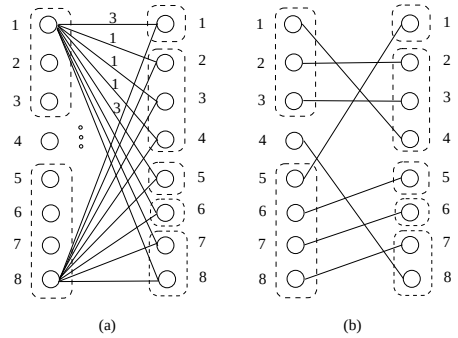


Figure 3. (a) The edge weighted complete bipartite graph and (b) the optimal solution.

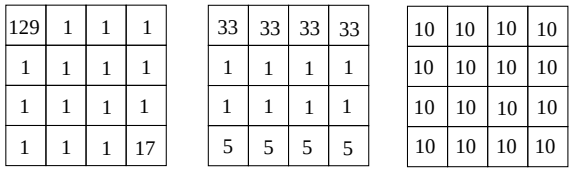
A naive approach to solve the matching problem is to enumerate all m perfect matchings and find an optimal one among them. A better solution called Hungarian method[1] exists. The following is the algebraic formulation for the matching problem. We let x_{ij}, $(i,j = 1,\ldots,m)$, be a set of variables. m is the number of nodes in the node sets of the complete bipartite graph $B = (V,U,E)$, where V, U are two node sets, and E is the edge set. $x_{ij} = 1$ means that the edge (v_i,u_j) is included in the matching, whereas $x_{ij} = 0$ means not. An optimal solution is to:

$$ \begin{array}{ll} \text{Minimize} & \Sigma_{ij} c_{ij} x_{ij} \\ \text{subject to} & \sum_{j=1} x_{ij} = 1 \quad i = 1,\ldots,m \\ & \sum_{i=1} x_{ij} = 1 \quad j = 1,\ldots,m \end{array} \quad (5) $$

With this definition, the bipartite graph problem is converted into a matrix problem. The rows of the matrix x represent the nodes in V, and the columns represent the nodes in U. The value of entry c_{ij} is the cost of assigning node v_i to node u_j.

There are several polynomial implementations for the Hungarian method. Our implementation is based on Munkres' [13], which describes the manual manipulation of a two-dimensional matrix by starring and priming zeros and by covering and uncovering rows and columns. Another implementation [14] solves the problem in $O(m^3)$. This implementation applies the solution to the max-flow problem with some modifications. A corresponding flow network G can be defined for the bipartite graph B, introducing two new nodes s and t. $2m$ edges are added in the graph, m from s to every node in V and m from every node in U to node t. In this way, the maximum flow problem can be explored.

To use the Hungarian method to load balancing in WSNs, we need to first convert the 2-D mesh to a complete bipartite graph using the follow procedure:



(a) (b) (c)

Figure 4. SMART(l): total cost 384.

1. Calculate the global average $\bar{v}$ and determine the "give", "take", and "neutral" state of each grid.

2. A node and edge weighted bipartite graph is constructed, where "give" and "take" grids appear at the left and right hand sides of the graph, respectively. The node weight corresponds to amount of overload and underload, and the edge weight represents the distance between the "give" and "take" grids in a matching pair.

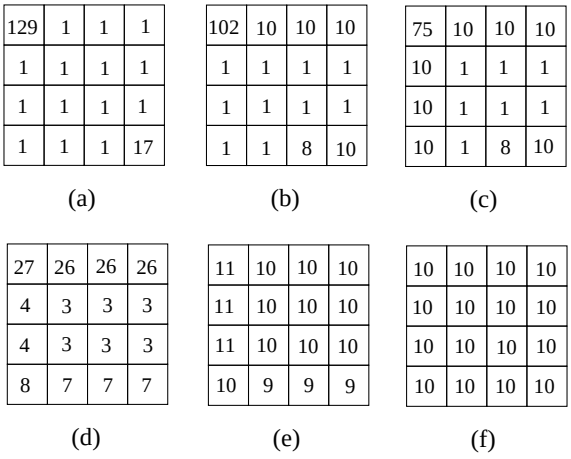
3. An edge weighted perfect bipartite graph is derived by expanding each node with weight k to k "clone" nodes. The edge weight of clone nodes will inherit from the original nodes.

Again, we use Figure 1 to illustrate the procedure. The global average in case is 5. There are three overloaded nodes and five underloaded nodes. $M[3,3] = 3$ means overloaded by 3 units and $M[1,2] = 1$ is underloaded by 1 unit. The edge weight is the Manhattan distance[2] between two end nodes $M[i,j]$ and $M[i^{'},j^{'}]$. That is, $\Delta x + \Delta y = |i - i^{'}| + |j - j^{'}|$. For example, the edge connecting $M[3,3]$ to $M[1,2]$ has a weight of 3. In Figure 2, the node and edge weighted bipartite graph shows weights of all edges connecting $M[3,3]$ to underloaded nodes.

In Figure 3 (a), the edge weighted complete bipartite graph of Figure 2 is shown, where each node (overloaded or underloaded) with weight k has k "clone" nodes. For example, $M[3,3]$ has three clone nodes labeled from 1 to 3. The Hungarian method is then applied to Figure 3 (a) and the optimal result is shown in Figure 3 (b). The optimal result shows that $M[5,5]$ (now with four clone nodes) needs to move one sensor to each of $M[1,2]$, $M[5,2]$, $M[2,3]$, and $M[4,3]$.

Suppose a BS (base station) is connected to the WSN, it can act as the central controller to collect all information from all leaders (clusterheads), execute the optimal algorithm, and then inform all leaders about the sensor movement from the current location to the destination location.

Instead of direct communication between each leader and the BS, some spanning-tree-based approaches can be applied. In WSNs, the BS is available only as an application frontend rather than as a centralized coordinator for coordinating basic network activities, including movement-assisted sensor deployment. Therefore, solutions based on local or limited global (such as prefix sum in the scan-based method) are more desirable.



(a) (b) (c)

(d) (e) (f)

Figure 5. SMART(g): total cost 352.

4 Extended Scan-Based Solutions

4.1 Threshold-based scan methods

In the original SMART, an "aggressive" approach is used where a local "give" state in a row or column can be a global "take" state (as in Example 1). To avoid this situation, a "conservative" approach can be used to decide local "give" and "take" state based on global average.

Again, we denote w_i as the number of sensors in grid i, and v_i the prefix sum of the first i grids in a row (or column) in the positive direction, i.e., $v_i = \sum_{j=1}^{i} w_j$. $v_n = \sum_{j=1}^{n} w_j$ is the total sum in the row (or column). Another negative direction prefix sum is exploited, where $v_i' = \sum_{j=i}^{n} w_j$, and $v_1' = \sum_{j=1}^{n} w_j$ is also the total sum in the row (or column). The negative prefix sum is achieved in the negative sweep where the average is sending out. Now, $\overline{w_l} = v_n/n$ is the average number of sensors in a local balanced state with respect to the current row (or column). $v = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}$ is the global total sum. Then $\overline{w_g} = v/n^2$ is the average number of sensors in a global balanced state. We define $\overline{w_m} = |\overline{w_g} - \overline{w_l}|/2$ as the mean of global and local balanced state. This approach is a compromise between conservative and aggressive approaches.

The proposed threshold-based scan method differs from the original SMART in its definition of threshold $\overline{w}$ used to determine the "give/take" state. Still, when $w_i - \overline{w} = 0$, grid i is in the "neutral" state. When $w_i - \overline{w} > 0$, it is overloaded and in the "give" state; and when $w_i - \overline{w} < 0$, it is underloaded and in the "take" state. $\overline{w}$ can be one of

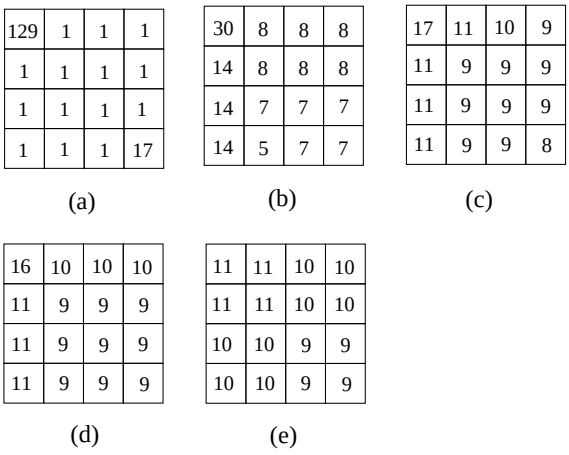


Figure 6. SMART($m, 3$): total cost 348.

three possible choices: $\overline{w_l}$, $\overline{w_g}$, and $\overline{w_m}$. Again, $\overline{v_i} = i\overline{w}$ is the the prefix sum in the balanced state under the given threshold $\overline{w}$, and $\overline{v_i}' = (n - i + 1)\overline{w}$ is that of the negative direction. $\overline{w}$ should be rounded to an integer.

In the original SMART, the threshold is based on the local average, $\overline{w_l}$, when "give" and "take" states are balanced in a row (or column). With a changing threshold, such a balance is no longer held. That is, there could be more "give" than "take" grids and vice versa. Therefore, $w_i^{\rightarrow}$ for load in the positive direction (or simply give-right) and ${}^{\leftarrow}w_i$ for load in the negative direction (give-left) are changed as follows: a grid is in "give" state if its value is over the given threshold $\overline{w}$. The amount of excessive load to be transferred to its right (or left) depends on the amount of underload to its right (or left) provided that amount does not cause the underload of the current node. More formally, we have

$$w_i^{\rightarrow} = \min\{w_i - \overline{w}, \max\{\overline{v'_{i+1}} - v'_{i+1}, 0\}\} \quad (6)$$

$${}^{\leftarrow}w_i = \min\{(w_i - \overline{w}) - w_i^{\rightarrow}, \max\{(\overline{v_{i-1}} - v_{i-1}), 0\}\} \quad (7)$$

The following steps are used in the proposed threshold-based scan:

1. If $\overline{w} \neq \overline{w_l}$, determine global balanced value $\overline{w_g}$.
2. Perform a row scan followed by a column scan using the selected $\overline{w}$.
3. If $\overline{w} \neq \overline{w_l}$, repeat step (2) using $\overline{w} = \overline{w_l}$.

$\overline{w_g}$ in step (1) can be calculated during step (2). Basically, $\overline{w_g}$ is determined after row and then column scans. However, in these scans there are no actual sensor movements. Movements occur once $\overline{w}$ is derived from $\overline{w_g}$. Step (3) is needed since the result of step (2) cannot guarantee a globally balanced state. When $\overline{w} = \overline{w_m}$, one variation of the algorithm is to repeat step (2) a constant (c) number of times before applying step (3).

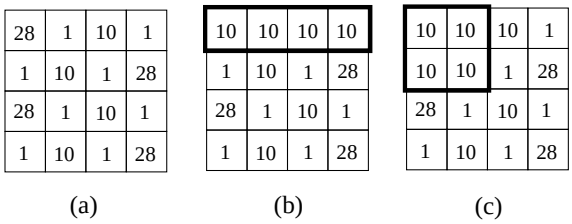


Figure 7. (a) initial network, (b) first step of SMART(g), (c) first step of H-SMART(g).

To simplify the notion, we use SMART(g), SMART(l), and SMART(m, c) to represent the threshold-based scan that uses global average, local average (the original SMART), and mean of global and local average, respectively. c in SMART(m, c) corresponds to the number of iterations of step (2). When c is 1, SMART(m, c) is simply written as SMART(m).

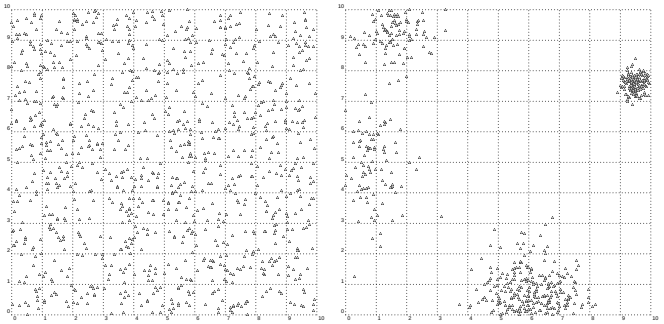
Since the Hungarian method is a global method, it can be done in one round. As mentioned above, SMART(l) can be done in two rounds, which means one row scan and one column scan. SMART(g) needs 4 rounds. One row scan, one column scan, and two rounds in step (3), which can be viewed as applying SMART(l) here. SMART(m, c) needs $2c + 2$ rounds. We will provide the proper c value in the simulation, which is quite small. Note that the traditional diffusion method [15] requires a large number of iterations to converge. Figures 4, 5, and 6 are working procedures of SMART(l), SMART(g), and SMART($m, 3$) applied on a sample 4×4 mesh.

4.2 Hierarchical-based scan methods

In hierarchical-based scan methods, the 2-D mesh is partitioned into four submeshes in a recursive way. The row and then column scans are applied to each submesh in a bottom-up fashion.

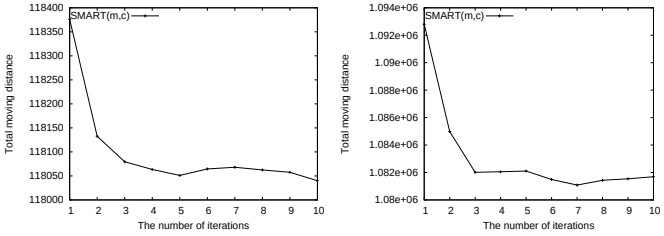
Suppose the 2-D mesh is a $2^k \times 2^k$ mesh (called a level-k mesh) and is partitioned into four $2^{k-1} \times 2^{k-1}$ submeshes. Each submesh is then recursively partitioned until the original 2-D mesh is partitioned into 2^{2d} $2^{k-d} \times 2^{k-d}$ submeshes and 2^{k-d} is sufficiently small. The following steps are used in the proposed threshold-based scan:

1. If $\overline{w} \neq \overline{w_l}$, determine global balanced value $\overline{w_g}$.
2. For $i = 0$ to d, perform a row scan followed by a column scan using the selected $\overline{w}$ on $2^{k-d-i} \times 2^{k-d-i}$ submeshes.



(a) Random (b) Multiple-cluster ($p = 4$)

Figure 8. Different initial sensor distribution.



(a) Random distribution (b) Multiple-cluster distribution

Figure 9. Moving distance of SMART(m, c) with difference iteration number c ($n = 10, m = 500$).

3. If $\overline{w} \neq \overline{w_l}$, use $\overline{w} = \overline{w_l}$ to perform a row scan followed by a column scan on the $2^k \times 2^k$ mesh.

This hierarchical approach is another heuristic approach where the excessive load in a "give" state is more likely to be moved to a nearby "take" state than to another "take" state. In this way, cases like Example 2 will be reduced. H-SMART needs $d + 2$ iterations for a $2^k \times 2^k$ mesh. In the first $d + 1$ iterations, the selected $\overline{w}$ can be $\overline{w}_g$, $\overline{w}_l$, or $\overline{w}_m$, and 2 rounds are needed for each iteration. In the last iteration, one row scan and one column scan (2 rounds) are executed to ensure a globally balanced state. Therefore, H-SMART needs $2(d + 1) + 2$ rounds totally. Figure 7 shows H-SMART applied to a sample 4×4 mesh.

5 Simulation

5.1 Simulation environment

We set up the simulation in a $5,000 \times 5,000$ monitoring area. We use three kinds of distributions as the initial deployment of sensors. The first is random distribution where sensors are randomly deployed in the entire area. The second is one-cluster distribution, where the sensors follow a normal distribution to form one clustered area. The third is multiple-cluster distribution, where sensors are deployed to form several clustered areas of different sizes. Figure 8 shows samples of the initial distribution ($m = 500$).

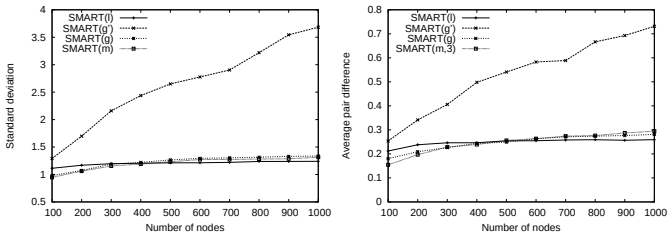
The tunable parameters are as follows. (1) The number of grids $n \times n$. We use 16 as the value of n, when H-SMART is analyzed, and 10 in the rest of the simulation. (2) The number of sensors m. We vary m's value from 100 to 1000. When n is 16, m varies from 256 to 1280 with the step 256. (3) The normal distribution parameter σ in one-cluster distribution. σ is the standard deviation of the normal distribution for the initial deployment. When σ is 10, around 50% of the sensors are in a 50% region of the area. When σ is 1, around 98% of the sensors are in a 10% region of the area. (4) The number of sensor clusters in multiple-cluster distribution p. We vary p from 1 to 10. For each sensor cluster, the normal distribution parameter is randomly selected from 1 to 10. (5) The number of iterations in H-SMART c. We use simulation to find a proper value for c.

The performance metrics are (a) deployment quality and (b) cost. Deployment quality is shown by the balance degree measured by the standard deviation of the number of sensors in all the grids, and also the average difference between every pair of grids. Deployment cost is measured by the energy consumption, in terms of overall moving distance and also, to a less extent, the number of total moves. Since the number of rounds, which represents the convergence rate of the algorithms, are static except SMART(m, c), we only test the round number of SMART(m, c), and find the proper c for it.

5.2 Simulation results

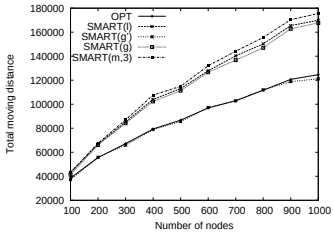
Figure 9 shows the resultant performance of SMART(m, c) with different c in both random and multiple-cluster distributions. The total moving distance decreases with the growth of c. This is because SMART(m) balances the distribution to the median of global average and local average, and after one iteration, the local average changes and the new median is generated for further balancing. Thus, more iterations lead to a more balanced state and when SMART(l) is applied, as in step (3), to achieve the final balanced state, the moving distance is smaller. The performance does not change much after three iterations. Therefore, we use 3 as the value of c in the following simulation.

Figure 10 illustrates the performance of the optimal solution (OPT), and the distributed solutions including SMART(l), SMART(g), and SMART(m) in random distribution. To check the effect of step (3), we simulate SMART(g'), which is SMART(g) without step (3). (a) shows the resultant standard deviation. SMART(g') has a

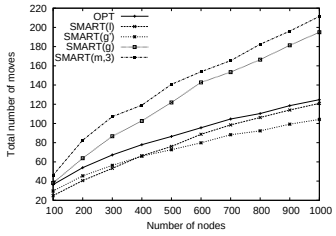


(a) Balance degree

(b) Pair difference

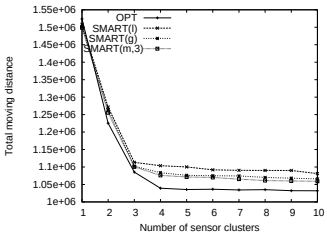


(c) Moving distance

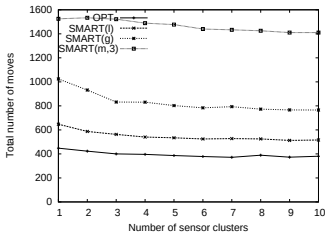


(d) Number of moves

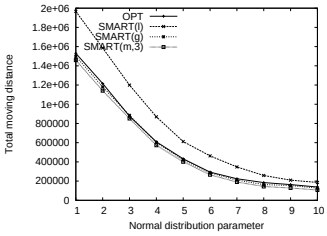
Figure 10. Random distribution ($n = 10$).



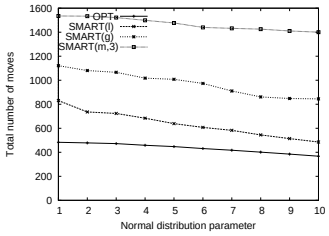
(a) Moving distance (M)



(b) Number of moves (M)



(c) Moving distance (O)



(d) Number of moves (O)

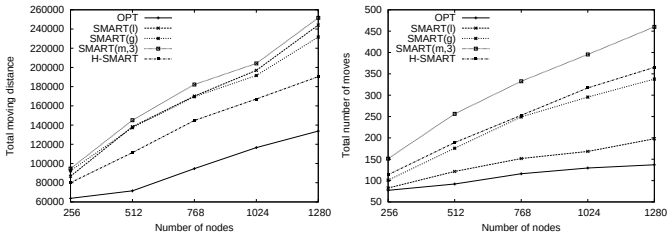
Figure 11. Multiple-cluster (M)/One-cluster (O) normal distribution ($n = 10, m = 500$).

large standard deviation while SMART(l), SMART(g) and SMART(m) have smaller ones. The standard deviation of OPT is 0 (not shown in the figure). (b) shows average pair difference. The results are similar with (a). Since SMART(l) is applied to both algorithms, the desired final balance degree is guaranteed since the difference between the maximum and minimum load grids is upper bounded by 2, we do not examine the performance of balance degree in the following simulation. (c) and (d) show the moving distance and number of moves, respectively. SMART(m) has the most moving distance, while SMART(g) has a smaller moving distance than SMART(l). OPT has the smallest moving distance. SMART(m) has the most number of moves. SMART(g) has the second largest number of moves. SMART(l) has an even smaller number of moves than OPT. Because SMART(l) can not completely balance the distribution.

Figure 11 shows the performance in multiple-cluster and one-cluster distribution. (a) and (b) are the moving distance and the number of moves in multiple-cluster. With the growth of the number of clusters, the total moving distance decreases and the number of moves decreases slightly. This is because the distribution tends to be balanced with more sensor clusters. SMART(g) and SMART(m) have smaller moving distances than SMART(l). SMART(m) has the smallest among the three. The numbers of moves are the opposite. SMART(m) has the largest while SMART(l) the smallest. OPT has the best performance in both moving distance and number of moves. (c) and (d) are results in one-cluster distribution. With the growth of σ, the moving distance decreases and the number of moves decreases slightly. This is because, when σ is large, the distribution is close to random distribution. OPT has larger moving distance than SMART(l) because SMART(l) does not balance the loads as it does under the one-cluster distribution circumstance. The relative performance of SMART(l), SMART(g), and SMART(m) are similar with that of multiple-cluster distribution. However, in one-cluster distribution, the moving distances of SMART(g) and SMART(m) are close to OPT.

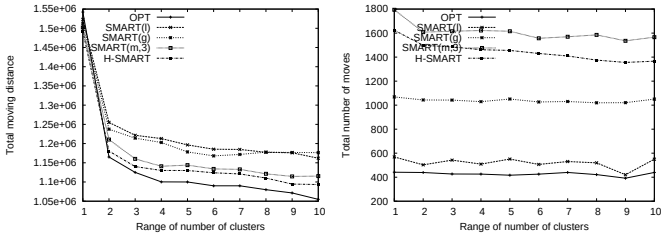
Figure 12 is the performance comparison of H-SMART with other algorithms. We use 16 as the value of n. The number of nodes in random distribution varies from 256 to 1280, with the step 256. Both the moving distance and the number of moves of all the algorithms are larger than those when n is 10, because having more grids makes the final distribution more balanced, which needs more consumption. In H-SMART, SMART(g) is used as the fundamental operation, because it has the best overall performance except OPT. (a) and (b) show the moving distance and the number of moves in random distribution with the growth of m. We can see that H-SMART further reduces the moving distance of SMART(m), and its number of moves is between those of SMART(g) and SMART(l). (c) and (d) are results of multiple-cluster distribution. H-SMART has better performance than SMART(m) in both the moving distance and the number of moves. (e) and (f) are results from one-cluster distribution. These results are consistent with those of $n = 10$, and H-SMART further increases the performance of SMART(m).

Simulation results can be summarized as follows: (1) The optimal solution has the best overall performance. (2)



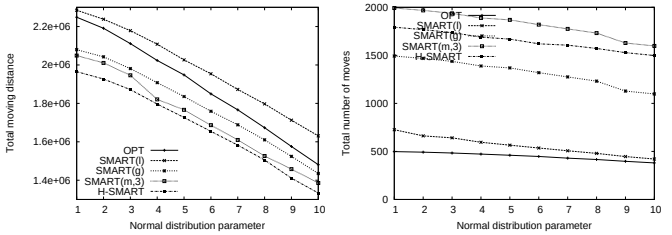
(a) Moving distance in random distribution

(b) Number of moves in random distribution



(c) Moving distance in multiple-cluster distribution

(d) Number of moves in multiple-cluster distribution



(e) Moving distance in one-cluster distribution

(f) Number of moves in one-cluster distribution

Figure 12. Comparison of H-SMART with other algorithms ($n = 16$, $m = 512$).

In all distributions, SMART(g) and SMART(m) have larger numbers of moves than that of SMART(l). (3) The iteration number of SMART(m) is as small as 3 to achieve a stable performance. (4) H-SMART further reduces the moving distance of SMART(m) without improving of the number of moves in most distributions.

6 Conclusions

We present an optimal solution to the movement-assisted sensor deployment problem using global network information. We also consider several heuristics without global information based on SMART. One is a scan-based approach, and the other one is a hierarchical-based approach. The simulation results show that the optimal solution achieves best overall performance. Among the local solutions, the hierarchical-based algorithm has the best performance, and these extended SMART algorithms have better performance than the original SMART in total moving distance, especially in one-cluster distribution, where its total moving distance is as low as that of the optimal solution.

References

[1] J. Wu and S. Yang, "SMART: A scan-based movement-assisted sensor deployment method in wireless sensor networks," in *Proceedings of INFOCOM*, 2005.

[2] G. T. Sibley, M. H. Rahimi, and G. S. Sukhatme, "Robomote: A tiny mobile robot platform for large-scale sensor networks," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[3] A. Howard, M. J. Mataric, and G. S. Sukhatme, "An incremental self-deployment algorithm for mobile sensor networks," *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, September 2002.

[4] O. Khatib, "Real time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, August 1986.

[5] G. Wang, G. Cao, and T. La Porta, "Movement-assisted sensor deployment," in *Proceedings of INFOCOM*, March 2004.

[6] G. Wang, G. Cao, T. La Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *Proceedings of INFOCOM*, 2005.

[7] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," in *Proceedings of INFOCOM*, March 2003.

[8] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *Proceedings of MobiCOM*, 2002.

[9] Y. Xu, J. Heidemann, and D. Estrin, "Geography informed energy conservation for ad hoc routing," in *ACM/IEEE International conference on Mobile Computing and Networking*, 2001.

[10] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan, and K. K. Saluja, "Sensor deployment strategy for target detection," in *Proceedings of WSNA*, 2002.

[11] M. Locateli and U. Raber, "Packing equal circles in a square: a deterministic global optimization approach," *Discrete Applied Mathematics*, vol. 122, pp. 139–166, Octobor 2002.

[12] D. Du, F. Hwang, and S. Fortune, "Voronoi diagrams and delaunay triangulations," *Euclidean Geometry and Computers*, 1992.

[13] "Dictionary of algorithms and data structures," 2005, http://www.nist.gov/dads/HTML/munkresAssignment.html.

[14] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization, algorithms and complexity*, Dover publications, INC, 1998.

[15] E. Luque, A. Ripoll, A. Cortes, and T. Margalef, "A distributed diffusion method for dynamic load balancing on parallel computers," in *Proceedings of 3rd Euromicro Workshop on Parallel and Distributed Processing*, 1995.