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
Email: jie@cse.fau.edu, syang1@fau.edu

***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 balance, where each unit of monitoring area is covered by the same number of sensors. The movement-assisted sensor deployment deals with moving sensors to meet coverage and load balance requirements. In SMART [1], various optimization problems are defined to minimize different parameters, including total moving distance, total number of moves, communication/computation cost, and convergence rate. In this paper, we focus on minimizing total moving distance and propose an optimal, but centralized solution, based on the Hungarian method. This solution is illustrated in an application where the monitoring area is a 2-D grid-based mesh. 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 distributed solutions.**

***Index Terms*— Dimension exchange, Hungarian method, load balance, scan, sensor coverage, sensor deployment, wireless sensor networks.**

I. Introduction

Wireless sensor networks (WSNs) [2], [3] combine processing, sensing, and communications to form a distributed system capable of self-organizing, self-regulating, and self-repairing. The efficiency of a sensor network depends on the coverage of the monitoring area. Although, in general, a sufficient number of sensors are used to ensure a certain degree of redundancy in coverage so that sensors can rotate between active and sleep modes, a good sensor deployment is still needed to balance the workload of sensors. By load balance, we mean each unit of monitoring area is covered by the same number of sensors.

This work was supported in part by NSF grants CCR 0329741, CNS 0422762, CNS 0434533, ANI 0073736, and EIA 0130806.

In general, two methods can be used to enhance the coverage: *incremental sensor deployment* and *movement-assisted sensor deployment*. Incremental self-deployment [4] 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 [5], on the other hand, uses a potential-field-based approach to move existing sensors by treating sensors as virtual particles, subject to virtual forces. Some extended virtual force methods are proposed in [6], [7] and [8], which are based on disk packing theory [9] and the virtual force field concept from robotics [4]. Basically, the movement-assisted sensor deployment deals with moving sensors to ensure coverage and then load balance if needed. Note that here load balance implies coverage and hence it is a stronger requirement. To achieve coverage (and load balance), various optimization problems can be defined to minimize different parameters, including total moving distance, total number of moves, communication/computation cost, and convergence rate.

The amounts of *cost* and *delay* are usually used as measures of all schemes for achieving a balanced state. The cost consists of three components: the mechanical movement of each sensor, computation of each sensor, and the electronic communication of each sensor. The cost of mechanical movement is dominant and can be measured by the total moving distance and, to a lesser extent, the number of moves. The electronic communication depends on both the number of transmissions and the size of message in each transmission. Computation is generally minimal unless a sophisticated computation process is used. Delay is measured as the time (steps) needed to achieve a balanced state.

In SMART [1], Wu and Yang related the sensor deployment to the load balance problem in parallel processing and pointed out their differences. For example, in WSNs, both the moving distance and the number of moves are important because of relatively heavy energy consumption to start or stop a move. They proposed a *scan-based solution* that does not resort to global (load) information. This solution can achieve load balance and minimize total moving distance of sensors in 1-D arrays. One unique issue in WSNs called the communication hole problem was also addressed.

In this paper, we focus on load balance solutions in WSN that minimize total moving distance of sensors. The monitoring area is a 2-D grid-based mesh (simply called a 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 total moving distance.

The contributions of this paper are as follows:

1) We systematically discuss the drawback of existing movement-assisted sensor deployment in WSNs.
2) We propose an optimal load balance solution based on the classic Hungarian method that achieves minimum total moving distance.
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 [10] and GAF [11] 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 point (cluster) has at least one sensor. Each grid point has one leader (clusterhead) to coordinate activities with leaders of four neighbors.

The remainder of the paper is organized as follows: Section 2 reviews some existing methods on movement-assisted sensor deployment, including SMART, and related load balance approaches in parallel processing. Section 3 proposes an optimal solution based on global information. Section 4 presents several extended scan-based solutions aiming to minimizing the total moving distance. Simulation results are presented in Section 5, and the paper concludes in Section 6.

II. Preliminaries and Related Work

We first review some related work on general load balance schemes, followed by an overview of existing work on movement-assisted sensor deployment with a focus on the scan-based approach proposed in SMART.

A. General load balance schemes

General load balance algorithms can be classified as local (such as iterative nearest neighbor exchanging [12]) and global (such as direct mapping [13]). The global approach relies on global information which is usually not scalable. Local algorithms can be either deterministic or stochastic. Diffusion and dimension exchange are two widely used local deterministic methods based on iterative nearest neighbor exchange.

In the diffusion method [14], the balancing procedure is divided into a sequence of synchronous steps. At each step, each node interacts and exchanges load with all its neighbors. In the dimension exchange method [15], the edges of the graph are colored such that no two adjacent edges have the same color. A "dimension" is then defined as a collection of edges with the same color. At each iteration, one particular color (dimension) is considered and every two adjacent nodes i and j connected by an edge with the selected color exchange and balance their load. Both methods are iterative and are based on nearest neighbor exchange. Although no information on load distribution is needed in local methods, iterative methods incur a significant number of rounds (moves in WSNs).

B. Movement-assisted sensor deployment

The sensor placement issue has been widely studied recently [16], [17], [18]. Random placement of sensors may not satisfy the deployment requirement due to the hostile deployment environment. Two methods can be used to enhance the coverage: incremental sensor deployment and movement-assisted sensor deployment.

In incremental sensor deployment [4], 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. Most existing movement-assisted sensor deployment protocols rely on the notion of virtual force to move existing sensors from an initial unbalanced state to a balanced state. These protocols are similar to nearest neighbor exchanging in load balancing. Sensors are involved in a sequence of computation (for a new position) and movement.

In [8], Zou and Chakrabarty proposed a centralized virtual force based mobile sensor deployment algorithm (VFA), which combines the idea of potential field and disk packing [9]. 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.

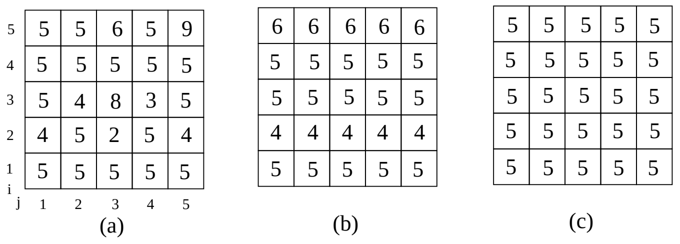
In [6], Wang, Cao, and La Porta developed a novel distributed self-deployment protocol for mobile sensors. They used Voronoi diagrams [19] to find coverage holes in the sensor network, and proposed three algorithms, VEC (Vector-based), VOR (Voronoi-based), and Minimax, to guide sensor movement toward 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 balance.

In [7], Wang, Cao, and La Porta further explored the motion capability of sensors for relocation to deal with sensor failure or respond to new events. The algorithm contains two phases. The first one is redundant sensor location, and the second is redundant sensor relocation. A grid-quorum solution was proposed to quickly locate the closest redundant sensors to the target area, where a sensor failure occurs. This solution uses the concept of quorum to locate sensors with low message complexity. Then a cascaded movement scheme was developed to relocate the located redundant sensors in a timely, efficient, and balanced way.

C. SMART: a scan-based approach

In SMART, a scan-based approach, a hybrid of local and global is adopted. 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.

A typical scan operation [20] involves a binary operator $\oplus$ and an ordered set $[w_0, w_1, ..., w_{n-1}]$, where each w_i represents the number of sensors in a grid, and returns the ordered set $[w_0, (w_0 \oplus w_1), ..., (w_0 \oplus w_1 \oplus, ..., \oplus w_n)]$. In SMART, only integer addition and boolean AND operations are used for scan. Using integer addition, the scan operation will return partial and total sums of the number of sensors of a 1-D array, which is a row or column of the given 2-D mesh. Since each grid position and n are known, average load information can be easily calculated and distributed as can the overload/underload situation of each ordered subset corresponding to a prefix of the ordered set.



(a)

i \ j	1	2	3	4	5
5	5	5	6	5	9
4	5	5	5	5	5
3	5	4	8	3	5
2	4	5	2	5	4
1	5	5	5	5	5

(b)

6	6	6	6	6
5	5	5	5	5
5	5	5	5	5
4	4	4	4	4
5	5	5	5	5

(c)

5	5	5	5	5
5	5	5	5	5
5	5	5	5	5
5	5	5	5	5
5	5	5	5	5

Fig. 1. An ideal case for SMART: (a) initial deployment, (b) after row scan, and (c) after column scan.

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. Within each row and column, the scan operation is used to calculate the average load and then to determine the amount of overload and underload in grids. Load is shifted from overloaded grids to underloaded grids in an optimal way to achieve a balanced state.

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 direction of the first sweep is called *positive* (with increasing order of grid ID) and that of the second sweep *negative*. 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

TABLE I

THE SCAN PROCESS ON THE THIRD ROW OF FIGURE 1.

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

$w_i - \overline{w} < 0$, 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 1 (c), respectively. Table I shows details of the row scan on the third row where i is the column number. 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 "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. In 2-D meshes, although the total number of moves is bounded by a factor of 2 compared with the optimal number of moves, the total moving distance is unbounded.

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.

III. AN OPTIMAL SOLUTION

This section starts with an optimal solution based on the classic Hungarian method. Several possible centralized implementations of this method in WSNs are then discussed. Finally, potential drawbacks of this approach are outlined.

A. Hungarian method

Let us consider the *edge weighted matching problem* in a complete bipartite graph $K_{m,m}$ with numbers associated edges called 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 maximum (or minimum).

For a maximization problem, consider the auction problem where each of m bidders offers a price for each of m products. Suppose each bidder will get one and only one product. The auctioneer wishes to assign a product to each bidder to maximize the total profit. For a minimization problem, consider a building project requires several buildings to be built simultaneously. A number of contractors place their offers. We wish to assign contractors to buildings so that the total cost of getting the buildings done is minimized.

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, E is 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.

[1]In honor of the Hungarian mathematicians D. Kőnig and E. Egerváry who developed it.

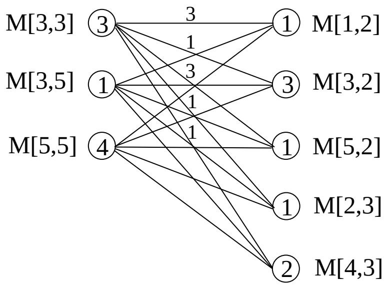


Fig. 2. The node and edge weighted bipartite graph of Figure 1 with "give" grids at the left-hand side and "take" grids at the right-hand side.

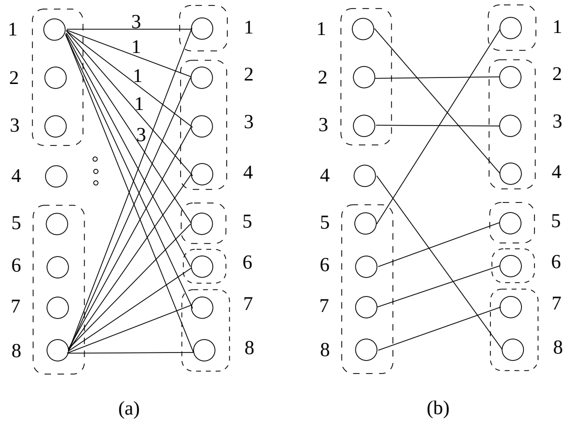


Fig. 3. (a) The edge weighted complete bipartite graph of Figure 2 and (b) the optimal solution.

There are several polynomial implementations for the Hungarian method. Our implementation is based on Munkres' [21], which describes the manual manipulation of a two-dimensional matrix by starring and priming zeros and by covering and uncovering rows and columns. In this method, the smallest entry in each row/column is subtracted from all the entries of that row/column to generate zero entries without changing the optimal solution. Then lines are drawn through each row or column so that all the zero entries of the matrix are covered and a minimum number of such lines has been used. If the number of covering lines is m, an optimal assignment of zeros is possible. Otherwise, the smallest entry not covered by any line is subtracted from all the entries in columns not covered by a covering column, and added to all entries in rows covered by a covering row. This step can be repeated until the optimal covering is found. Since the total cost of the matrix decreases with every step, the optimal assignment of zeros can be found in a finite number of steps.

Another implementation [22] solves the problem in $O(m^3)$. This implementation applies the solution to 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 balance in WSNs, we need to first convert the 2-D mesh to a complete bipartite graph using the follow procedure:

1) Calculate the global average $\bar{v}$ and determine "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) A 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 between two end nodes $M[i,j]$ and $M[i',j']$. That is, $\Delta x + \Delta y = |i - i'| + |j - j'|$[2]. 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 $M[4,3]$. The optimal matching of Figure 1 is also shown in the matrix of Figure 4 as in the Munkres' implementation where selected edges are boxed.

Note that the above method can easily be extended to cases where sensor can be moved to its diagonal neighbors (four in all). In this case, the edge weight is changed

[2]The general distance between two points is defined as $((\Delta x)^k + (\Delta y)^k)^{1/k}$. When $k = 2$, it is Euclidian distance, and when $k = 1$, it is Manhattan distance.

$$\begin{array}{cc}
\begin{array}{c}8\\7\\6\\5\\4\\3\\2\\1\end{array} & \begin{pmatrix}
7 & 5 & 5 & 5 & 3 & 5 & 3 & \boxed{3} \\
7 & 5 & 5 & 5 & 3 & 5 & \boxed{3} & 3 \\
7 & 5 & 5 & 5 & 3 & \boxed{5} & 3 & 3 \\
7 & 5 & 5 & 5 & \boxed{3} & 5 & 3 & 3 \\
\boxed{5} & 3 & 3 & 3 & 5 & 3 & 3 & 3 \\
3 & 1 & 1 & \boxed{1} & 3 & 1 & 1 & 1 \\
3 & 1 & \boxed{1} & 1 & 3 & 1 & 1 & 1 \\
3 & \boxed{1} & 1 & 1 & 3 & 1 & 1 & 1
\end{pmatrix} \\
 & \begin{array}{cccccccc}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{array}
\end{array}$$

Fig. 4. (a) The matrix form of the edge weighted complete bipartite graph of Figure 3 with the minimum total cost 22.

to $\min\{\Delta x, \Delta y\} + |\Delta y - \Delta x| = \max\{\Delta x, \Delta y\}$. The Hungarian method can be applied to WSNs with various shapes of monitoring areas. In fact, it can also be used for optimal matching in WSNs where the monitoring area is a set of discrete targets [23].

The cost of implementing the Hungarian method for load balance in WSNs is $O(m^3)$, where m is the amount of overloads (underloads) which is bounded by the number of sensors. Usually, the number of sensors is one or two magnitudes higher than the number of grids (n). However, the main problem of the Hungarian method lies in its centralized implementation as will be discussed in the next subsection.

B. Implementations

The solution based on the Hungarian method is centralized, which is costly to implement in general in WSNs. Here are some possible implementations.

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. For example, the BS can broadcast its intent. With the regular topology of 2-D meshes, broadcast can be implemented efficiently without resorting to blind flooding. However, information aggregation is needed at each branch of the broadcast tree, although some optimization methods can be used to construct a "balanced" tree in 2-D meshes with a minimum weight of the maximum branch [24].

In WSNs, the (remote) 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) is more desirable. However, an optimal solution without using global information does not seem to be possible. In the next section, we will look at several extended scan-based solutions where each sensor has limited memory storage capacity.

IV. Extended Scan-Based Solutions

We consider two types of extension to the scan: *threshold-based* and *hierarchical-based*. In threshold-based scan methods, "give" and "take" states depend not only on the local average in a row/column but also on the global average. In hierarchical-based scan methods, the 2-D mesh is partitioned into four submeshes level by level and the scan-based method is applied in a bottom-up fashion.

A. 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.

We first introduce some new notations. 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 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 $\overline{v_i}' = (n-i+1)\overline{w}$

129	1	1	1
1	1	1	1
1	1	1	1
1	1	1	17

(a)

33	33	33	33
1	1	1	1
1	1	1	1
5	5	5	5

(b)

10	10	10	10
10	10	10	10
10	10	10	10
10	10	10	10

(c)

Fig. 5. A 4×4 sample network, total cost 384 with SMART(l).

129	1	1	1
1	1	1	1
1	1	1	1
1	1	1	17

(a)

102	10	10	10
1	1	1	1
1	1	1	1
1	1	8	10

(b)

75	10	10	10
10	1	1	1
10	1	1	1
10	1	8	10

(c)

27	26	26	26
4	3	3	3
4	3	3	3
8	7	7	7

(d)

11	10	10	10
11	10	10	10
11	10	10	10
10	9	9	9

(e)

10	10	10	10
10	10	10	10
10	10	10	10
10	10	10	10

(f)

Fig. 6. A 4×4 sample network, total cost 352 with SMART(g).

129	1	1	1
1	1	1	1
1	1	1	1
1	1	1	17

(a)

30	8	8	8
14	8	8	8
14	7	7	7
14	5	7	7

(b)

17	11	10	9
11	9	9	9
11	9	9	9
11	9	9	8

(c)

16	10	10	10
11	9	9	9
11	9	9	9
11	9	9	9

(d)

11	11	10	10
11	11	10	10
10	10	9	9
10	10	9	9

(e)

Fig. 7. A 4×4 sample network, total cost 348 with SMART($m, 3$).

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

$$
\begin{aligned}
w_i^{\rightarrow} &= \min\{w_i - \overline{w}, \max\{\overline{v'_{i+1}} - v'_{i+1}, 0\}\} \quad (6)\\
{}^{\leftarrow}w_i &= \min\{(w_i - \overline{w}) - w_i^{\rightarrow},\\
&\qquad \max\{(\overline{v_{i-1}} - v_{i-1}), 0\}\} \quad (7)
\end{aligned}
$$

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).

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 requires a large number of iterations to converge.

Figures 5, 6, and 7 are working procedures of SMART(l), SMART(g), and SMART($m, 3$) applied on a sample 4×4 mesh. In Figure 5, the result after row scan is shown in (b), and column scan, in (c), the network is balanced with the total moving distance 384. In Figure 6, (b) and (c) are row and column scans using the global average. Then SMART(l) is applied in (d) and (e) as described in step (3). The resultant total moving distance is 352. Figure 7 is of SMART($m, 3$). (b) is the result of first round which contains a row and a column scan. (c) is second round, and (d) is the third round. Then SMART(l) is applied to achieve the balanced state. The

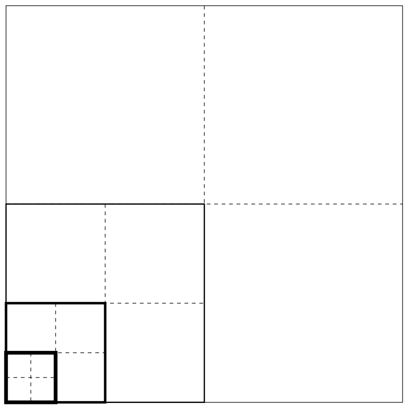


Fig. 8. A sample 4-level hierarchical partition.

total moving distance is 348. The total moving distance for this example using the Hungarian method is 350.

B. 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. Figure 8 shows such a 4-level hierarchical partition.

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.
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 other "take" state. In this way, cases like Example 2 will be reduced. A simple analysis is given in Appendix. 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 9 shows a sample 4×4 mesh. (b) shows the first step of SMART(g). We can see that the first row is balanced with grid $(1,1)$ contributes 9 nodes to grid $(1,2)$ and grid $(1,4)$ each. (c) is the first step of H-SMART, where grid $(1,1)$ contributes to grid $(1,2)$ and grid $(2,1)$. Grid $(1,4)$ will be covered by grid $(2,4)$ later. The total moving distance for this example are 144 and 72, for SMART(g) and H-SMART, respectively.

28	1	10	1
1	10	1	28
28	1	10	1
1	10	1	28

(a)

10	10	10	10
1	10	1	28
28	1	10	1
1	10	1	28

(b)

10	10	10	1
10	10	1	28
28	1	10	1
1	10	1	28

(c)

Fig. 9. An example. (a) is initial network, (b) is first step of SMART(g), (c) is first step of H-SMART(g).

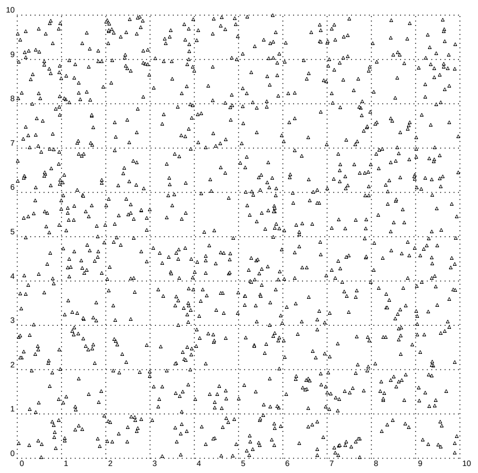
V. Simulation

In this section, we present the results of our simulation of the proposed optimal movement-assisted sensor deployment method, and various heuristic local methods.

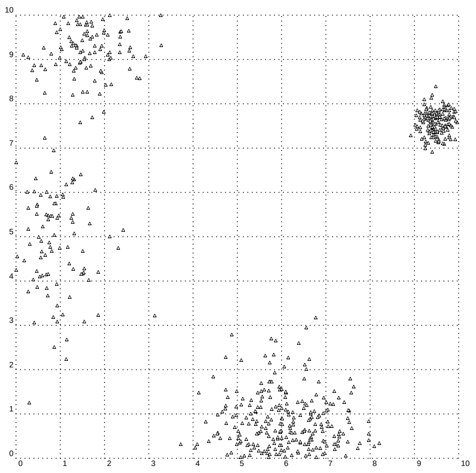
A. Simulation environment

All approaches are tested on a custom simulator. We set up the simulation in a $5,000 \times 5,000$ monitoring area, which is the target field. Sensors can be deployed in this area following a given distribution. We use three kinds of distribution 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 10 shows samples of the initial distribution. (a) is random distribution, and (b) is multiple-cluster distribution (4 sensor clusters). The tunable parameters in our simulation 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 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, which can control the density degree of the sensor clustering. We use 1 to 10 as its values. When σ is 10, around 50% sensors are in 50% region of the



(a) Random distribution



(b) Multiple-cluster distribution ($p = 4$)

Fig. 10. Samples of different initial sensor distribution ($m = 500$).

area. When σ is 1, around 98% sensors are in 10% region of the area.

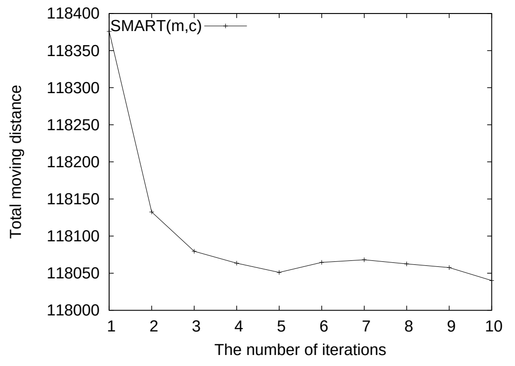
4) The number of sensor clusters in multiple-cluster distribution p. We vary p from 1 to 10. For each sensor cluster, the normal distribution parameter is randomly selected from 1 to 10.
5) The number of iterations in H-SMART algorithm c. We use simulation to find a proper value for c.

The performance metrics are (a) deployment quality and (b) cost. Deployment quality is shown by the balance degree measured by the standard deviation of sensor numbers in all the 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. The moving distance of sensors is proportional to the energy consumption and so is the number of moves. This is because sometimes, the startup of sensors may cause some additional cost. 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.

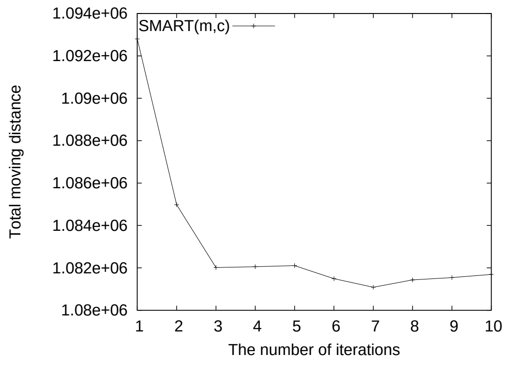
B. Simulation results

Figure 11 shows the resultant performance of SMART(m, c) with different c in both random and multiple-cluster distributions. We can see that in both 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 more balanced state and when SMART(l) is applied, as in step (3), to achieve final balanced state, the moving distance is smaller. We can also see that, the performance does not change much after three iterations. Therefore, we use 3 as the value of c in the following simulation.

Figure 12 illustrates the performance of the optimal solution (OPT), based on the Hungarian method, and its extensions, the distributed solutions in random distribution. The distributed solutions include SMART(l), SMART(g), and SMART(m). To check the effect of step (3), we simulate the SMART(g'), which is SMART(g) without step (3). (a) is the resultant standard deviation comparison of them, which represents the balance degree. We can see that SMART(g') has a very large standard deviation while SMART(l), SMART(g) and SMART(m) have relatively smaller ones. The standard deviation of OPT is 0 (not shown in the figure). (b) is the comparison of SMART(l), SMART(g), and SMART(m). We can see that these three have comparative performance. Since SMART(l) is applied to both algorithms, the desired final balance degree is guaranteed, we do not examine the performance of balance degree. Figures 12 (c) and (d) show the moving distance and number of moves in random distribution, respectively. We can see that SMART(m) has the most moving distance, while SMART(g) has smaller moving distance than SMART(l). OPT has the smallest moving distance. Although SMART(g') has a comparative moving distance with OPT, it will not be considered further since it cannot balance the distribution. 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 while OPT can.

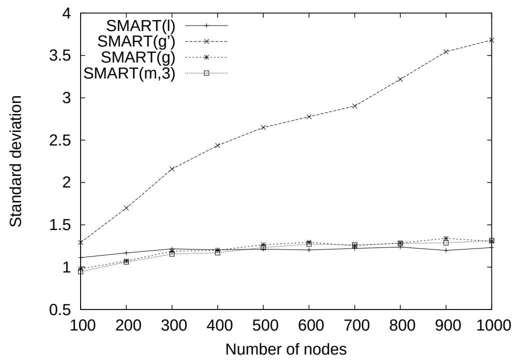


(a) Moving distance in random distribution

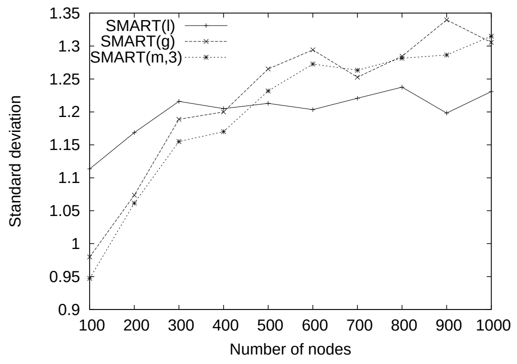


(b) Moving distance in multiple-cluster distribution

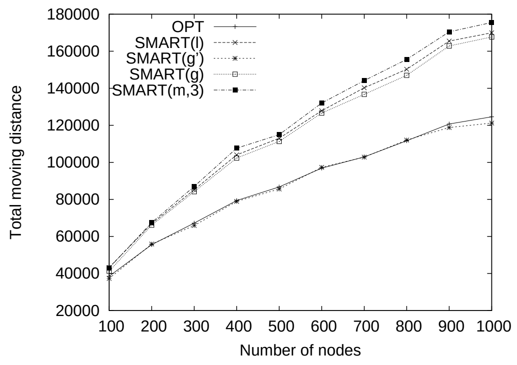
Fig. 11. SMART(m, c) with difference iteration number c ($n = 10, m = 500$).



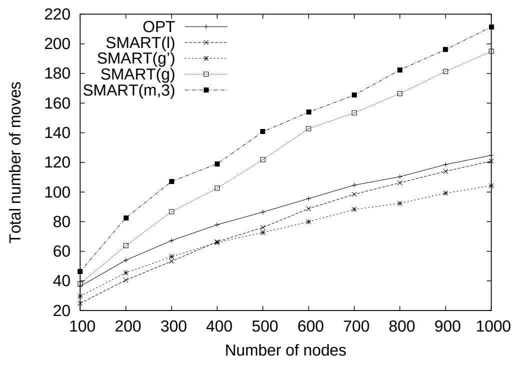
(a) Balance degree



(b) Balance degree



(c) Moving distance

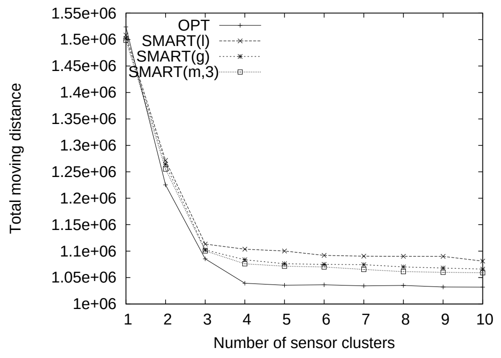


(d) Number of moves

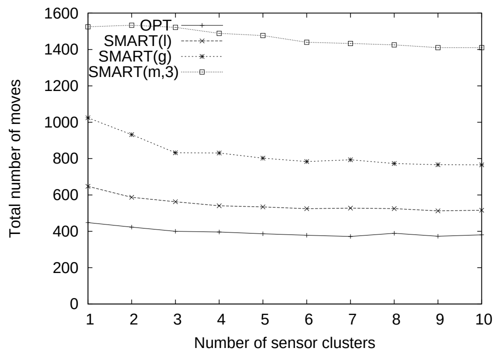
Fig. 12. Random distribution ($n = 10$).

Figure 13 shows the performance in multiple-cluster distribution. (a) is the moving distance, and (b) is the number of moves. We can see that 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 distance than SMART(l). SMART(m) has the smallest among the three. The numbers of moves show 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.

Figure 14 shows the performance in one-cluster distribution, where the normal distribution parameter σ varies from 1 to 10 to represent different sensor clustering degree. (a) is moving distance, and (b) is the number of moves. With the growth of σ, the moving distance decreases and the number of moves decreases slightly, as in Figure 13. This is because, when σ is large, the distribution is close to random distribution. The relative

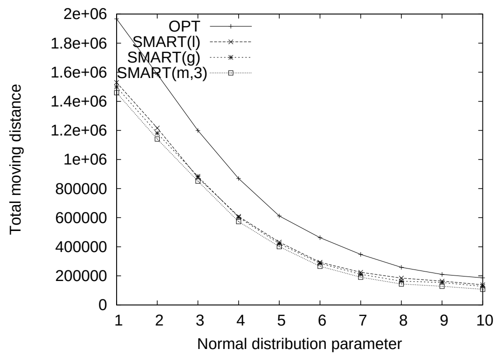


(a) Moving distance

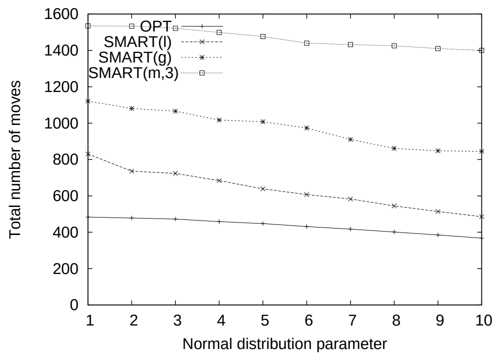


(b) Number of moves

Fig. 13. Multiple-cluster normal distribution ($n = 10, m = 500$).



(a) Moving distance



(b) Number of moves

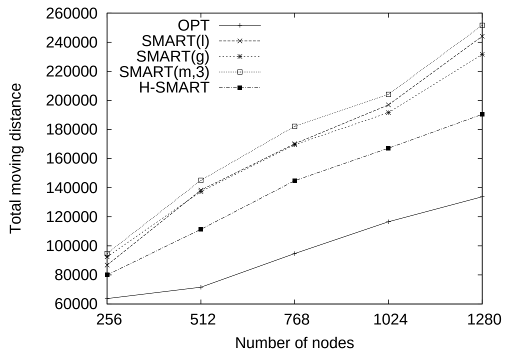
Fig. 14. One-cluster normal distribution ($n = 10, m = 500$).

performance of SMART(l), SMART(g), and SMART(m) in the term of moving distance and number of moves are similar with that of Figure 13. However, in one-cluster distribution, the moving distance of SMART(g) and SMART(m) is as small as that of OPT.

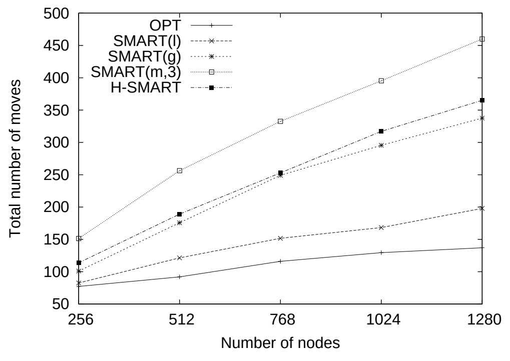
Figure 15 is the performance comparison of H-SMART with other algorithms. We use 16 as the value of n, thus H-SMART takes 4 levels. The number of nodes in random distribution varies from 256 to 1280, with the step 256. Both the moving distance and the number of moves of all the algorithms are larger than those when n is 10, because 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. 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 consist with those in Figure 14, and H-SMART further increases the performance of SMART(m) in both the moving distance and the number of moves.

Simulation results can be summarized as follows:

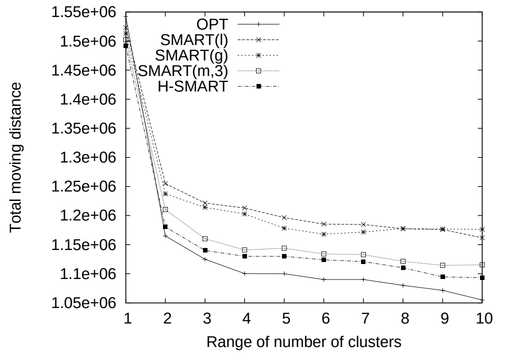
1) The optimal solution, which uses the Hungarian method, has the best performance in both the moving distance, the number of moves, and the balance degree in almost all kinds of initial distribution.
2) In one-cluster distribution, SMART(g) and SMART(m) have close moving distance to OPT, which is much smaller than that of SMART(l).
3) In multiple-cluster distribution, SMART(g) and SMART(m) have smaller moving distance than SMART(l), but still much larger than that of OPT.
4) In random distribution, SMART(g) has better moving distance than that of SMART(l), but SMART(m) has the largest moving distance.
5) In all distributions, SMART(g) and SMART(m) have larger numbers of moves than that of SMART(l). This is because they employ a step-



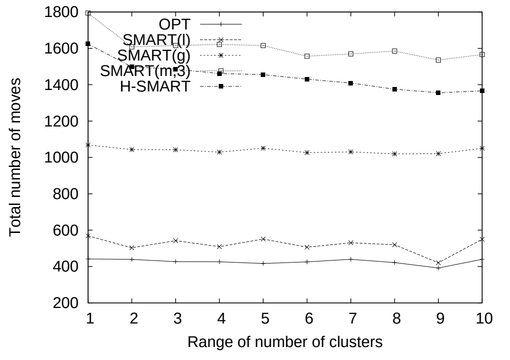
(a) Moving distance in random distribution



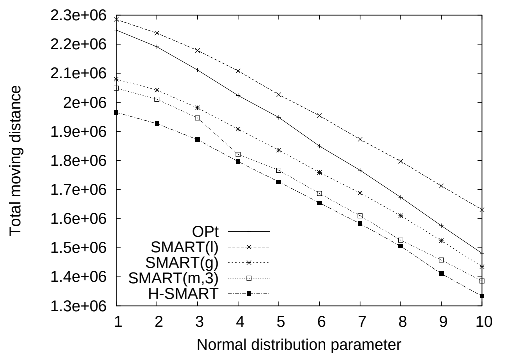
(b) Number of moves in random distribution



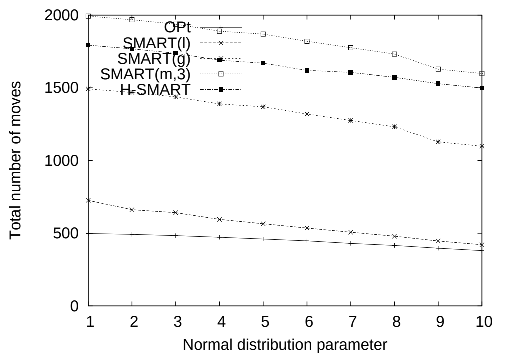
(c) Moving distance in multiple-cluster distribution



(d) Number of moves in multiple-cluster distribution



(e) Moving distance in one-cluster distribution



(f) Number of moves in one-cluster distribution

Fig. 15. Comparison of H-SMART with other algorithms ($n = 16$, $m = 512$ except (a) and (b)).

by-step movement to avoid excessive movement.

6) The iteration number of SMART(m) is as small as 3 to achieve a stable performance.
7) H-SMART further reduces the moving distance of SMART(m) without improving of the number of moves in most distributions.

VI. Conclusions

We present in this paper an optimal solution to the movement-assisted sensor deployment problem. This solution is implemented using global network information. We also consider several heuristics without global information. One is based on extending SMART, a scan-based approach, and the other one is a hierarchical-based method. The simulation results show that the optimal solution achieves best performance in all kinds of initial distributions. Among the local solutions, the hierarchical-based algorithm has the best performance, and the 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. In our future work, we will develop other movement-assisted sensor deployment methods, which are local and can achieve a performance close to the optimal one.

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] I. F. Akyildiz, W. Su, Y. Sankrasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communication Magazine*, pp. 102–114, August 2002.

[3] D. E. Culler and W. Hong, "Wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 30–33, June 2004.

[4] 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.

[5] 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.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] T. L. Casavant and J. G. Kuhl, "A communication finite automata approach to modeling distributed computation and its application to distributed decision-making," *IEEE Transactions on Computers*, vol. 39, no. 5, pp. 628–639, May 1990.

[13] H. C. Lin and C. S. Raghavendra, "A dynamic load balancing policy with a central job dispatcher (lbc)," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 148–158, February 1992.

[14] 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.

[15] H. Rim, J. Jang, and S. Kim, "An efficient dynamic load balancing using the dimension exchange method for balancing of quantized loads on hypercube multiprocessors," in *Proceedings of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, 1999.

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

[17] S. Dhillon, K. Chakrabarty, and S. Iyengar, "Sensor placement for grid coverage under imprecise detections," in *Proceedings of International Conference on Information Fusion*, 2002.

[18] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak, "Exposure in wireless ad-hoc sensor networks," in *Proceedings of Mobicom*, 2001.

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

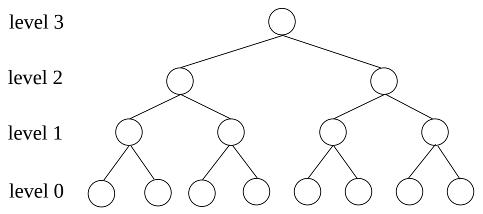


Fig. 16. A 4-level full tree as a 1-D projection of Figure 8.

[20] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1526–1538, November 1989.

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

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

[23] M. Cardei, J. Wu, M. Lu, and M. Pervaiz, "Maximum network lifetime in wireless sensor networks with adjustable sensing ranges," in *Proceedings of IEEE WiMob*, 2005.

[24] T.-S. Chen, Y.-C. Tseng, and J.-P. Sheu, "Balanced spanning trees in complete and incomplete star graphs," *IEEE Transaction on Parallel Distributed System*, vol. 7, no. 7, pp. 717–723, 1996.

Appendix

In H-SMART, we say a submesh covers a "give" grid and a "take" grid if both grids are within the submesh. We show that this smallest submesh in H-SMART is related to the Manhattan distance between these two grids.

Let $M[i,j]$ and $M[i^{'},j^{'}]$ be a pair of "give" and "take" grids. Δ is defined as $\Delta x + \Delta y = |i - i^{'}| + |j - j^{'}|$. Consider the smallest submesh covering a $\Delta \times \Delta$ square. To simplify the discussion, we assume $\Delta = 2^k$. By placing the square to every positive position of the mesh, it is easy to see that the probability of the smallest coverage submesh being a level-k submesh is 1/2, a level-$(k-1)$ submesh is 1/4, and so on. In fact, we can consider a projection of $\Delta \times \Delta$ square on x-axis (or y-axis). The problem becomes placing a line (of the projection) to every position at leaves of a full binary tree shown in Figure 16. In this figure, each node in tree corresponds to a square of Figure 8. Each child node is a submesh of the submesh that corresponds to its parent. Therefore, the expected size of the smallest coverage submesh is $2^{k+1} \times 2^{k+1}$. Since $\Delta/2 \leq \max\{\Delta x, \Delta y\} \leq \Delta$, the expected size of the smallest submesh that covers the "give" and "take" grids is between $2^k \times 2^k$ and $2^{k+1} \times 2^{k+1}$. When $\Delta \neq 2^k$, suppose $2^k < \max\{\Delta x, \Delta y\} < 2^{k+1}$, the expected value for coverage $\Delta x \times \Delta y$ rectangle is bounded between the ones for square $2^k \times 2^k$ and square $2^{k+1} \times 2^{k+1}$.